The shortest distance between
two points is under construction.
(Noelie Altito)

Chapter 3

Feature-based Motion I: Point-Correspondences

Algorithms for motion estimation can be coarsely divided into feature-based techniques and dense estimation techniques. The feature-based techniques are known to perform better with large motions while dense estimation techniques provide a higher accuracy. To combine the advantages, we integrated both approaches into one motion-estimation system. We describe the feature-based motion estimator in this and the following chapter while dense estimation is covered in Chapter 5. A feature-based motion estimator comprises three steps: detection of feature-points, establishing feature-correspondences, and estimation of camera motion parameters. In this chapter, we give an introduction to feature-based motion estimation and we cover the first two steps. We provide a survey of feature-point detectors and evaluate their accuracy. After that, we present a fast algorithm for determining correspondences between two sets of feature-points. The remaining parameter estimation step will be discussed in the next chapter.

3.1 Introduction

Video sequences generally comprise two types of motion: the motion of objects visible in the scene and the global motion that is caused by the moving camera. Object motion is usually also difficult to describe, since in general, objects can show articulated or deformable motion. In general, the motion of objects can be rather complex and very difficult to model. Examples are water waves, explosions, traffic lights, or other sudden changes. On the other hand, camera motion is restricted to only a few degrees of freedom like the camera rotation angles or its focal length. In the previous chapter, camera motion was described using a geometric model with a small set of parameters. When analyzing a video sequence, the inverse problem occurs: find the parameters which describe the apparent motion in the video sequence.

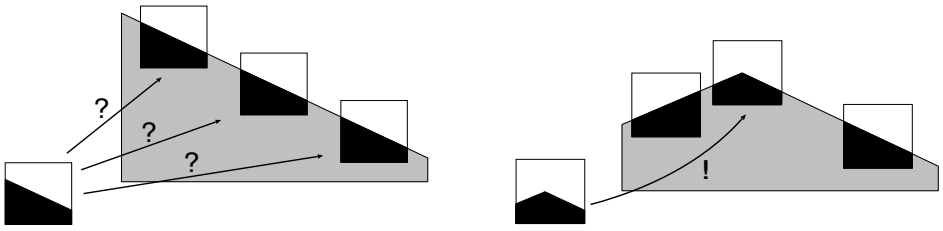
Techniques for estimating the motion parameters generally follow one of two fundamental approaches: *direct estimation* algorithms and *feature-based* algorithms.

- In the *direct estimation* algorithms, the motion parameters $\mathbf{H}$ are computed by minimizing the motion-compensated image difference between the two images I_t and I_{t+1}, thus computing

$$\min_{\mathbf{H}} \sum_{\mathbf{p}} |I_t(\mathbf{p}) - I_{t+1}(\mathbf{H}\mathbf{p})|^2. \tag{3.1}$$

  If the motion model is simple, like the translatorial motion model of MPEG-2, the motion parameters can be found by an exhaustive search through the parameter space. However, for the projective motion model, there are eight free parameters, which raises the need for fast optimization algorithms. Usually, gradient-descent algorithms are applied to solve Equation (3.1). When initialized with a good set of parameters, the direct estimation methods can achieve high accuracy. On the other hand, convergence of the gradient-descent algorithms requires a good initialization, especially if the motion between images is large.

- The second approach for motion estimation are *feature-based* techniques. They determine a small set of feature points in each of the input frames and establish correspondences between matching points. The feature points are selected such that the motion of the point can be computed with high reliability and accuracy. After the point-correspondences have been established, the motion parameters are determined by fitting the motion model to the point-correspondence data.



(a) Aperture problem along a line.

(b) No aperture problem at corners.

Figure 3.1: *(a) A small area with a linear structure cannot be located precisely in a larger image. (b) The aperture problem does not occur if a corner or any other texture that varies in two dimensions is visible in the window.*

Since the feature-based methods primarily use the position of feature-points instead of a direct comparison of the image data, the feature-based methods are more robust to changes of illumination or noise than the direct methods. Furthermore, they allow large motions between images and they are also faster to compute. On the other hand, their estimation accuracy is generally below that of dense estimation algorithms. To combine the respective advantages of both approaches, it is possible to use the result of the feature-based approach as an initialization of the successive dense estimation step. Since the initialization is already close to the correct motion, the gradient-descent algorithm will converge to the correct minimum. Without this close initialization, the algorithm may not find this minimum if the motion is too fast.

Our segmentation system also follows this approach of combining a feature-based initialization of the motion model with a refinement of accuracy in a succeeding direct estimation step. We discuss feature-based motion estimation in this chapter and the direct estimation algorithms in the successive chapter.

3.1.1 Basics of feature-based motion estimation

Estimating image motion is an ill-posed problem in many situations. For small image patches or even single pixels, there is usually not enough information available to determine the motion reliably. Consider for example a small image patch with a straight region border (Fig. 3.1(a)). If we take this small piece and try to find the matching position in a larger picture, we see that this is impossible, since the pattern can be found all along the

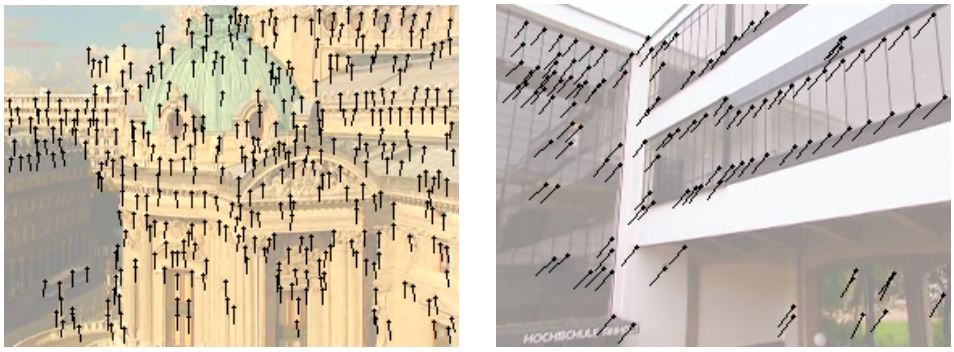


Figure 3.2: *(a) Example of detected interest-points and their correspondence in the previous frame. (b) Example of a repetitive image texture. Correspondences cannot be established easily, since many similar interest-points exist.*

region border. The problem can already be identified by only examining the search pattern. In our case, the pattern is only structured in one direction, namely perpendicular to the line direction. Along the line, the texture is uniform, making it impossible of finding a best-matching position. The problem becomes even worse if the pattern only shows uniform color, providing no directed structure in the texture at all. This undeterminism is known as the *aperture* problem. It is only possible to determine the position of the pattern reliably if the pattern shows variations in two directions. This is the case, for example, at corners (see Fig. 3.1(b)). Here, the position of the pattern can be determined precisely in both dimensions.

We have seen that for the purpose of motion estimation, little information can be obtained from areas of uniform or texture that is structured only in one-dimension. Hence, the idea of feature-based methods is to concentrate only on a small set of *interest-points*, for which the motion can be determined reliably. An example of detected interest-points and the respective motion vector to the corresponding point in the previous frame is shown in Figure 3.1.1.

The decision whether a position in the image is a good feature-point is generally made as a local decision based only on the image content in the neighborhood of the interest-point. However, on a more global view, it can still be difficult to determine the motion for an interest-point, even when it shows a clear corner. If the image content shows a repetitive structure (like seen in Figure 3.1.1), many interest-points with comparable a neighborhood exist and it is not clear which points corresponds to each other.

Resolving this problem is the task of a second algorithm, which com-

putes the actual correspondences between the two sets of interest-points. A variety of algorithms have been developed for this problem and they can be distinguished as algorithms that only consider the positions of the interest-points, and algorithms that also take the image content around each interest-point into account. The second approach has the advantage that the image content can guide the matching algorithm. On the other hand, this approach does not work well if the transformation between both images is so large that the image content cannot be easily compared. Similarity metrics that are invariant to e.g. affine transformations have been proposed in the literature, but since the motion between frames in a video sequence is usually small, we will not consider them further.

3.1.2 From feature-points to motion parameters

Once we obtained a set of corresponding points between two images, we can use this information to determine the motion parameters. Since each point-correspondence gives us two equations of constraints — one for the horizontal component and one for the vertical — we need four point-correspondences to solve for the eight parameters of the perspective motion model (see Figure 3.3). Inserting any four correspondences $\mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_i$ with $\mathbf{p}_i = (x_i, y_i, 1)$, $\hat{\mathbf{p}}_i = (\hat{x}_i, \hat{y}_i, 1)$ into the inhomogeneous formulation from Eq. (2.10) and multiplying with the denominator results in the linear equation system

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1\hat{x}_1 & -y_1\hat{x}_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1\hat{y}_1 & -y_1\hat{y}_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2\hat{x}_2 & -y_2\hat{x}_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2\hat{y}_2 & -y_2\hat{y}_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3\hat{x}_3 & -y_3\hat{x}_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3\hat{y}_3 & -y_3\hat{y}_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4\hat{x}_4 & -y_4\hat{x}_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4\hat{y}_4 & -y_4\hat{y}_4 \end{bmatrix} \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{pmatrix} = \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \\ \hat{x}_2 \\ \hat{y}_2 \\ \hat{x}_3 \\ \hat{y}_3 \\ \hat{x}_4 \\ \hat{y}_4 \end{pmatrix}, \tag{3.2}$$

which can be easily solved for the model parameters h_{ik}[1]. While four correspondences are enough to solve for the motion parameters, we usually have many more correspondences available. However, these are contaminated with outliers and inaccuracies in the feature positions. Apart from errors in the computation of the point-correspondences, we also consider correspondences that are part of foreground object motion as outliers. To separate this mixed data into clean sets of compatible motion is the main topic of Chapter 4.

[1] See also Annex B for an alternative way to obtain the motion parameters with lower computation cost.

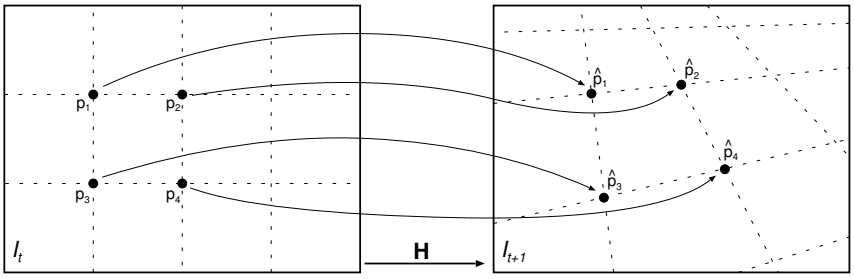


Figure 3.3: *Four point-correspondences between two images define the projective transformation.*

To summarize the typical feature-based motion estimation approach, we can conclude that a feature-based motion estimator comprises three main processing steps:

1. detection of interest-points,
2. establishing correspondences between two sets of interest-points, and
3. estimating the motion-model parameters for the dominant motion.

Algorithms for the first two steps are discussed and evaluated in the remainder of this chapter, while the third step is covered in the following chapter.

3.2 Interest-point detectors

In the past, a large number of interest-point detectors have been proposed and this topic is still an active area of research. One of the first interest-point detectors was the Moravec detector [129]. It is a simple *ad-hoc* algorithm to detect image locations that show variations in many different directions. Shi and Tomasi [167] describe an interest-point detector that is derived directly by identifying the sort of texture for which the motion estimation process is well defined.

Many authors propose to use interest-point detectors that are in fact corner-point detectors[2]. However, it is interesting to note that for our application, it is not important that the interest-points are placed at some

[2]Note that we use the three terms *feature*-point, *interest*-point, and *corner*-point almost synonymically. The term *feature*-point originates from the motion estimation area, where the generation process of these points is not considered. The terms *interest* and *corner* point are used for detection algorithms, depending on whether the algorithm is designed to identify just arbitrary *interest* points, or *corners* in the image.

specific position like the true corner position. As long as the detector consistently places the interest-points to the same corresponding positions in each image, its output is valuable.

A commonly used corner detector is the *Harris* (also known as *Plessey*) corner detector [84]. Interestingly, the Harris detector appears to be very similar to the Shi-Tomasi algorithm using only a different detection criterion on the same extracted data. A more recent corner detector that claims to have especially good corner localization capabilities is the SUSAN corner detector [170]. We will describe and evaluate the performance of the four mentioned corner detectors in the following sections.

Recent work concentrates on designing interest-point detectors that are invariant to some classes of image transformations. Since image transformations are usually small between successive frames of a video sequence, ordinary interest-point detectors work well for our application and we will not consider these invariant detectors here. For further information see, e.g., [126, 110].

Criteria for good detectors

Interest-points should be placed at positions where the image content around the feature allows to identify the corresponding position in a second image with high reliability. Placing a feature-point onto a line is not sufficient, since a line is a one-dimensional feature that only allows to fix the position perpendicular to the line. The position of the interest-point along the line, on the other hand, remains uncertain. Consequently, good features must show a change of texture in two directions so that they can be localized reliably. In practice, corners, T-junctions, or complex textures provide a good localization (see Fig. 3.4(a)-(c)). However, interest-points are only selected locally and an interest-point that seems easy to track at first sight can actually be a bad choice if there are ambiguities because of, e.g., a repetitive pattern in the image (Fig. 3.1.1). Another problem case can occur if the video sequence shows several objects moving in different directions. If two of these objects overlap, artificial T-junctions can be visible along the object boundary. If one object moves, this junctions may seem to move in a different direction than the objects (see Fig. 3.4(d)). Both problems can only be solved globally and thus, they are not considered in the feature-point detector.

For the successive processing steps, where we want to establish correspondences between points in pairs of images, two properties of the interest-point detector are especially important.

- First, for images which show basically the same content in a slightly

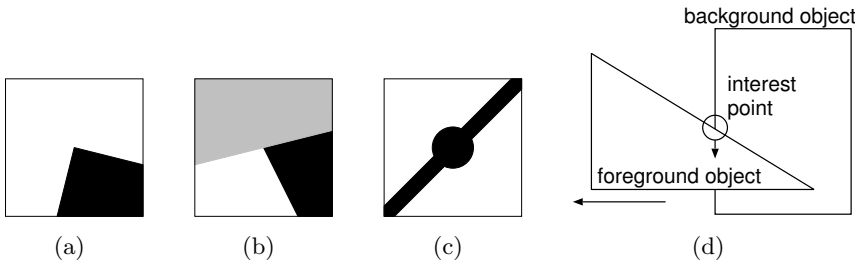


Figure 3.4: *(a)-(c) Suitable structures for placing of interest-points. (d) An artificial T-junction is visible at the boundary between two objects. If one objects moves, the T-junction moves virtually in a different direction.*

different view, the same features should be detected in both images. Features that are only found in one of the images cannot be part of a point-correspondence and would be useless for the succeeding steps. Even worse, these features will contaminate the data with noise and may lead to wrong correspondences which will then again make the robust estimation harder.

- A second criterion for a good interest-point detector is that the position of the interest-points should be invariant to the image transform that is present between the two images. Consider for example that the interest-point placement is biased to one direction, e.g., all points have some offset to the left. If we have a rotational transform between the two considered images, the interest-points would not lie at corresponding image positions anymore. This imposes problems in later steps where we want to estimate the image transforms based on the assumption that corresponding feature-points in a pair of images move according to the image transform. Clearly, a larger error in the localization of feature-points also increases the error in the estimation of the transform.

3.2.1 Moravec interest-point detector

The Moravec interest-point operator is based on the observation that the image content around a good feature-point should show variations in every possible direction. Following this principle, the Moravec operator considers a window $\mathcal{W}(x, y)$ of size $w \times w$ around a pixel (x, y) and calculates the sum of absolute differences between pairs of pixels that are either horizontal,

vertical, or diagonal neighbours. More specifically, the sums for the four directions are calculated by

$$S_{horiz}(x,y) = \sum_{(x',y')\in\mathcal{W}(x,y)} \left|I(x',y') - I(x'+1,y')\right|, \tag{3.3}$$

$$S_{verti}(x,y) = \sum_{(x',y')\in\mathcal{W}(x,y)} \left|I(x',y') - I(x',y'+1)\right|, \tag{3.4}$$

$$S_{diag1}(x,y) = \sum_{(x',y')\in\mathcal{W}(x,y)} \left|I(x',y') - I(x'+1,y'+1)\right|, \tag{3.5}$$

$$S_{diag2}(x,y) = \sum_{(x',y')\in\mathcal{W}(x,y)} \left|I(x',y') - I(x'-1,y'+1)\right|. \tag{3.6}$$

Since the differences along all directions must be large, the operator determines the minimum

$$S(x,y) = \min\{S_{horiz}(x,y), S_{verti}(x,y), S_{diag1}(x,y), S_{diag2}(x,y)\}, \tag{3.7}$$

it applies a local non-maximum suppression to these values, and it selects those positions as interest-points for which $S(x,y)$ exceeds a threshold τ_{mo}.

A disadvantage of the Moravec operator is that it is an anisotropic operator that is sensitive to image rotation. The reason for this is that only four directions are considered and that the diagonal pixel distances are a factor of $\sqrt{2}$ larger than the horizontal and vertical distances.

3.2.2 Shi-Tomasi detector

Instead of using an intuitive *ad-hoc* definition of an interest detector, Shi and Tomasi propose in [167] to define the interest-point detector by looking at the motion-estimation problem itself and finding conditions when this problem can be solved reliably. The interest-points they search for are not defined by intuitive terms like *corners*, but only by their ability to track the features reliably. They assume a translatorial motion $\mathbf{d}$ between two images I_t and I_{t+1}. If we consider again a small window $\mathcal{W}$ around the feature, the matching error E can be written as

$$E = \sum_{\mathbf{p}\in\mathcal{W}} (I_t(\mathbf{p}+\mathbf{d}) - I_{t+1}(\mathbf{p}))^2. \tag{3.8}$$

After approximating I_t by a linear Taylor expansion $I_t(\mathbf{p}+\mathbf{d}) \approx I_t(\mathbf{p}) + \nabla I_t^\top \mathbf{d}$, we can write the matching error as

$$E = \sum_{\mathbf{p}\in\mathcal{W}} \left(I_t(\mathbf{p}) - I_{t+1}(\mathbf{p}) + \nabla I_t^\top \mathbf{d}\right)^2. \tag{3.9}$$

To determine the motion $\mathbf{d}$, we find the minimum matching error by setting the derivatives to zero:

$$\frac{\partial E}{\partial \mathbf{d}} = 2 \sum_{\mathbf{p} \in \mathcal{W}} \left(I_t(\mathbf{p}) - I_{t+1}(\mathbf{p}) + \nabla I_t^\top \mathbf{d} \right) \nabla I_t = 0. \tag{3.10}$$

It follows that

$$\sum_{\mathbf{p} \in \mathcal{W}} (\nabla I_t \nabla I_t^\top) \cdot \mathbf{d} = \sum_{\mathbf{p} \in W} \nabla I_t \cdot (I_t(\mathbf{p}) - I_{t+1}(\mathbf{p})), \tag{3.11}$$

which is actually a linear equation system $\mathbf{G}\mathbf{d} = \mathbf{e}$, where

$$\mathbf{G} = \begin{bmatrix} \sum (\partial I_t(\mathbf{p})/\partial x)^2 & \sum (\partial I_t(\mathbf{p})/\partial x)(\partial I_t(\mathbf{p})/\partial y) \\ \sum (\partial I_t(\mathbf{p})/\partial x)(\partial I_t(\mathbf{p})/\partial y) & \sum (\partial I_t(\mathbf{p})/\partial y)^2 \end{bmatrix}, \tag{3.12}$$

$$\mathbf{e} = \begin{pmatrix} \sum (\partial I_t(\mathbf{p})/\partial x) \cdot (I_t(\mathbf{p}) - I_{t+1}(\mathbf{p})) \\ \sum (\partial I_t(\mathbf{p})/\partial y) \cdot (I_t(\mathbf{p}) - I_{t+1}(\mathbf{p})) \end{pmatrix} \tag{3.13}$$

and all the sums go over the window $\mathcal{W}$. To solve reliably for $\mathbf{d}$, the equation system should be well-conditioned. Shi and Tomasi claim that this is the case if the value of both Eigenvalues of $\mathbf{G}$ do not differ much and if the Eigenvalues exceed a minimum threshold. Speaking in terms of the image content, two small Eigenvalues correspond to an almost uniform content while two Eigenvalues with very different size indicate an edge-like content [96]. Consequently, the authors propose to use the smaller of the two eigenvalues λ_1, λ_2 as a detection criterion for feature-points. Since $\mathbf{G} = \{g_{ik}\}$ is a symmetric 2×2 matrix, the Eigenvalues can be computed easily as

$$\lambda_1, \lambda_2 = \frac{1}{2} \left(g_{00} + g_{11} \mp \sqrt{(g_{00} + g_{11})^2 - 4(g_{00} g_{11} - g_{01}^2)} \right). \tag{3.14}$$

Detection is carried out by taking $\lambda(x, y) = \min\{\lambda_1, \lambda_2\}$ for each image position and assuming a feature-point if $\lambda(x, y) > \tau_{st}$ and $\lambda(x, y)$ is a local maximum in the image.

Using a weighted window

In the previously described algorithm, all pixels within the window $\mathcal{W}$ are included equally in the calculation. However, this leads to the problem that the window $\mathcal{W}$ is placed such that the window maximizes the number of high-gradient pixels in the window, regardless of their position in the window. This induces that the detected interest-point position will be

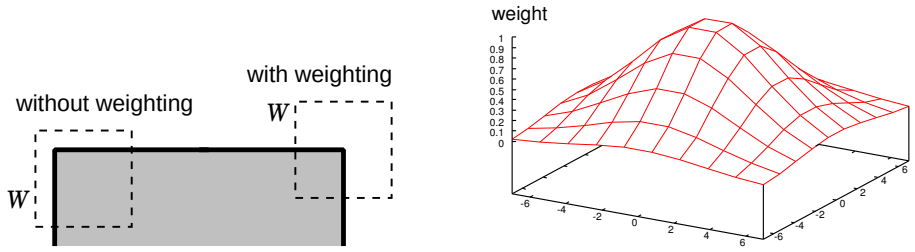
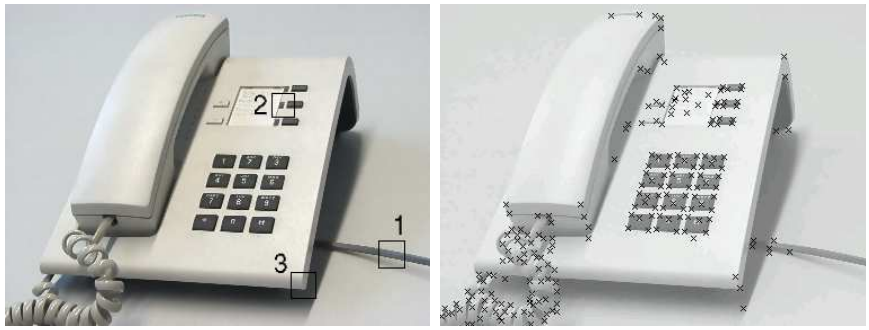


Figure 3.5: *Placement of the feature-point at a corner. (a) If all pixels within a window contribute equally (left window), the feature-point will not be located exactly at the corner, since the algorithm tries to maximize the amount of edge pixels within the window. When pixels at the center are weighted more than pixels further away (right window), the feature-point moves closer to the window center. (b) A Gaussian weighting function.*

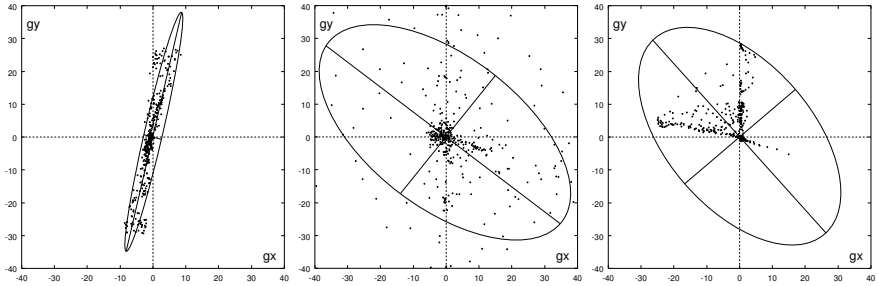
biased (see Fig. 3.5(a)). To prevent this effect, a weighting function w can be introduced to increase the weight of the center pixels. In our application, we use a cascade of five computationally efficient binomial filters on the gradient vector components to approximate a Gaussian windowing function (Fig. 3.5(b)).

3.2.3 Harris corner detector

The Harris corner detector [84] (also known as Plessey detector) is in fact very similar to the Shi-Tomasi operator, but in this section, we would like to derive the operator in a different way to give an alternative, more intuitive explanation of its function. We begin with examining the texture in a window $\mathcal{W}$ around the considered pixel position by observing the gradient vectors in this window. The idea is to classify the texture in the window based on the distribution of the gradient vectors. Figure 3.6 shows scatterplots of the gradient vectors for three example window locations. If we consider Window 1, which contains a linear edge, we obtain the gradient-vector distribution as shown in Figure 3.6(c). We see that all the gradient vectors have approximately the same orientation, where the major axis of the distribution is perpendicular to the image edge. For an image corner (Fig. 3.6(e)), different gradient orientations are present, each corresponding to one of the edges. This is comparable to complex texture content (Fig. 3.6(d)) where the gradient vectors are distributed more or less uni-



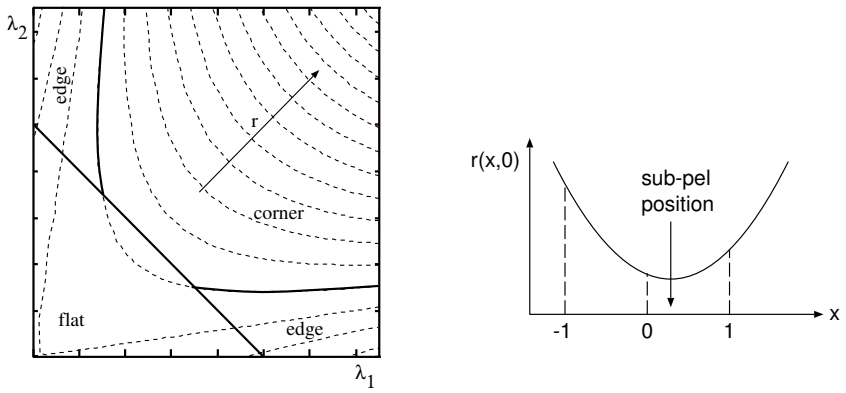
(a) Original image. (b) Detected feature-points.



(c) Window 1. (d) Window 2. (e) Window 3.

Figure 3.6: *(c)-(e) Scatter plots of gradient vectors for the indicated windows in (a). The detected corner positions are shown in (b).*

formly. Finally, for image content with only small luminance variations, all the gradient vectors are close to zero. To classify the different situations, we model the distribution of gradient vectors with a bivariate Gaussian distribution whose principal axes can be obtained as the principal components of the correlation matrix of the gradient vectors. Note that the correlation matrix is exactly equal to $\mathbf{G}$ as we defined it for the Shi-Tomasi operator in Eq. (3.12). Moreover, the length of the principal axes are again the Eigenvalues λ_1, λ_2 of the correlation matrix. If we look at the length of the principal axes for each of the texture types, we observe that two small axes indicate a flat image content, one large and one small axis indicates a linear structure, while two large axes indicate a corner or other complex texture. Following this observation, Harris and Stephens proposed the classification scheme of Figure 3.7(a) to classify a specific gradient-vector distribution into one of the classes *flat*, *edge*, and *corner*. According to this, a flat re-



(a) Harris operator: texture classification.

(b) Determining sub-pel corner position.

Figure 3.7: *(a) Classification of a pixel into one of the three classes* flat,edge,corner *is based on the two Eigenvalues* λ_1, λ_2 *of the correlation matrix. (b) The sub-pel accurate corner position is determined by fitting a quadratic function and taking their minimum position.*

gion is detected if $\lambda_1 + \lambda_2 < \tau_{ha}$, where τ_{ha} is a flat-region threshold. The decision function for the corner region is defined by

$$r = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = \mathrm{Det}(\mathbf{G}) - k \cdot \mathrm{Tr}(\mathbf{G})^2 \ > \ 0, \tag{3.15}$$

where $k \approx 0.06$. Note that the decision function is chosen such that it is not necessary to actually compute the Eigenvalues. Instead, it is sufficient to compute the matrix determinant and the matrix trace by exploiting their equivalence to the product and sum of the Eigenvalues, respectively. To locate the corner-points, local minima of r are determined and a corner is reported at this position if $r > 0$ and an additional flat area test ($\mathrm{Tr}(\mathbf{G}) \geq \tau_{ha}$, with τ_{ha} being a flat area threshold) is passed.

The Harris detector shows the same problem of biased corner-point placement as described earlier for the Shi-Tomasi detector. However, this problem can be easily approached in the same way by introducing pixel-weighting within the window. An example result of applying the Harris detector is shown in Figure 3.6(b).

Extension to sub-pel accuracy

The Harris corner detector described so far determines the location of corners with integer pixel accuracy. We have extended the algorithm by fitting a second-order polynomial to the horizontal and vertical neighborhood and taking the minimum of these polynomials as the sub-pixel position of the corner.

Assume that we found an (integer-accurate) local minimum of decision function r at $r(x_0, y_0)$. Since a translation does not affect the shape of the polynomial function, we can assume w.l.o.g. that $x_0 = y_0 = 0$. In the following, only the computation along the horizontal direction is described, since the computations for the vertical direction are equivalent. Inserting the values $r(-1,0), r(0,0), r(1,0)$ into the model of the polynomial as $ax^2 + bx + c = r(x,0)$ allows to easily compute the parameters as $a = (r(-1,0) + r(1,0))/2 - r(0,0)$, $b = (r(1,0) - r(-1,0))/2$, and $c = r(0,0)$. Computing the derivative and setting it to zero results in the position of the minimum

$$x = \frac{(r(-1,0) - r(1,0))/2}{r(-1,0) + r(1,0) - 2r(0,0)}, \tag{3.16}$$

which provides the sub-pel correction offset.

3.2.4 SUSAN corner detector

The principle of the SUSAN corner detector [170] is based on a morphology approach. While the previous detectors identify positions where the image content changes in differing directions, the SUSAN corner detector directly uses a definition of a corner in its algorithm. For this, it uses the USAN principle (*Univalue Segment Assimilating Nucleus*). The USAN is the area of a local neighborhood that has the same color as the center pixel of the considered neighborhood. This is illustrated in Figure 3.8 where the circles depict several examples of local neighborhoods.

In the cases where the circle center lies inside of the rectangle, the USAN consists of pixels from the rectangle. If the center is outside of the rectangle, the USAN only consists of pixels outside of the rectangle. When we calculate the size of the USAN area for every possible location, we observe that the USAN area is smallest at the corners (see Figure 3.9). This principle is now used in the SUSAN corner detector (SUSAN="*Smallest Univalue Segment Assimilating Nucleus*") by searching for local minima of the USAN area. Since the authors of [170] provide source code for the SUSAN corner detector, we integrated the original implementation into our experiments.

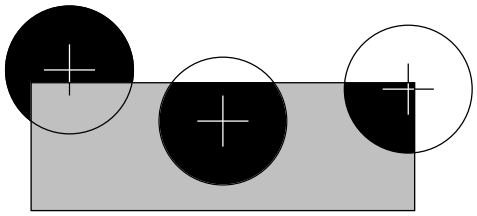


Figure 3.8: *USAN principle. The USAN is defined as the part of the neighborhood window that has the same color as the center pixel. The USAN area for different window positions is drawn black.*

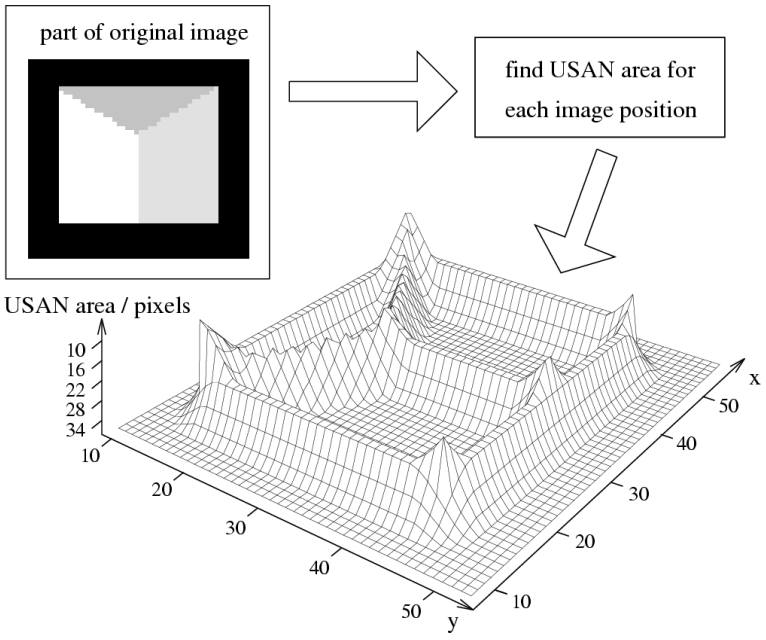


Figure 3.9: *SUSAN corner detector. The USAN area for a small sample image. Corners are detected by finding local minima of the USAN area. (Picture is taken from [170].)*

3.2.5 Evaluation

Reference correspondences

For the successive evaluations, we need ground-truth data that tells us which feature-point $\mathbf{p}_i$ in one frame corresponds to a feature-point $\hat{\mathbf{p}}_k$ in the second frame (written as $\mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_k$). Unfortunately, this information is not available easily and a manual generation of the data is impractical, since there are usually several hundred feature-points in each frame. As a solution, we assume that the transform $\mathbf{H}^\star$ between images is known. In our experiments, we used the output of our complete motion estimation system where we checked beforehand that the estimation result was very accurate. Based on this reference transformation $\mathbf{H}^\star$, we define a set of reference correspondences $\mathcal{R}$ which we use instead of ground truth data.

Let us denote the set of feature-points in the first image I_t as $\mathcal{F}_t = \{\mathbf{p}_i\}$ and as $\mathcal{F}_{t+1} = \{\hat{\mathbf{p}}_k\}$ for the second image I_{t+1}. We define the set of reference correspondences basically as $\mathcal{R}_\epsilon = \{\mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_k \mid d(\hat{\mathbf{p}}_k, \mathbf{H}^\star \cdot \mathbf{p}_i) < \epsilon\}$, but with the additional constraint that each feature-point is only included once. This set is constructed in a greedy approach where a pair of points is iteratively added to the set if their projection distance $d(\hat{\mathbf{p}}_k, \mathbf{H}^\star \cdot \mathbf{p}_i)$ is smallest and both points are not used yet in $\mathcal{R}_\epsilon$. The parameter ϵ defines the maximum displacement error that is allowed for a correpondence. This threshold is required, since the interest-point detectors can only estimate the point positions with some amount of error. Choosing a too low value for ϵ would result in missed feature-correspondences, whereas a too high value would include wrong matches. A good choice for ϵ will become apparent after we evaluate the repeatability of the interest-point detectors.

Evaluation criteria

We evaluate the performance of the described interest-point detectors according to two criteria:

- **Repeatability**: The fraction of detected interest-points in one frame that can also be found in the other frame. Features that are only found in one of the images cannot be part of a point-correspondence and consequently, they are useless for the succeeding motion analysis. Even worse, unmatched features would contaminate the data with noise and may lead to wrong correspondences, which would make the parameter estimation harder. To quantify this property, we define the *repeatability* of the interest-point detector as the fraction of detected

feature-points in one frame that can also be found in the other frame:

$$repeatability = \frac{|\mathcal{R}_\epsilon|}{\min(|\mathcal{F}_t|, |\mathcal{F}_{t+1}|)}. \tag{3.17}$$

This is the number of available correspondences, normalized by the minimum number of features in the two input frames. Since we cannot get more correspondences than the minimum number of features, the maximum value for *repeatability* is 1.0.

- **Accuracy**: A good interest-point detector provides unbiased positions that are invariant to the image transform that occurs between the two images. Clearly, a more consistent placement of feature-points leads to a higher accuracy of the motion estimation parameters, since the displacement errors are smaller. Note that there is no direct definition what the ideal position for an feature-point is, but the position should not jitter between frames.

  Assuming that we know the ground-truth transformation $\mathbf{H}^\star$ between the two frames, we define the accuracy of an interest-point detector as the mean distance between the feature-point position $\hat{\mathbf{p}}_\mathbf{k}$ in the second frame and the position of the corresponding feature-point in the first frame, mapped onto the second frame as $\mathbf{H}^\star \cdot \mathbf{p_i}$. Feature-points which have no counterpart in the other image are not considered. This gives the definition of the accuracy as

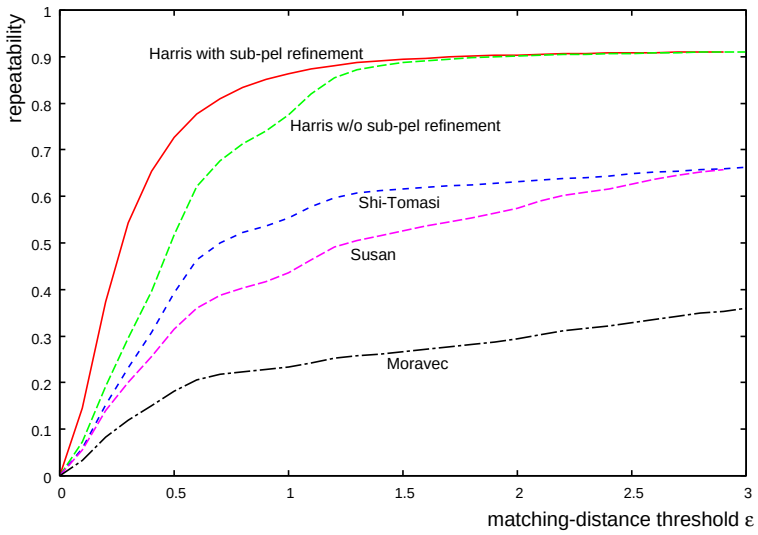
$$accuracy = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{p_i} \leftrightarrow \hat{\mathbf{p}}_\mathbf{k} \in \mathcal{R}} d(\hat{\mathbf{p}}_k \ , \ \mathbf{H}^\star \cdot \mathbf{p}_i). \tag{3.18}$$

  Note that a lower accuracy value indicates a better accuracy. Optimal accuracy is reached for $accuracy = 0$.

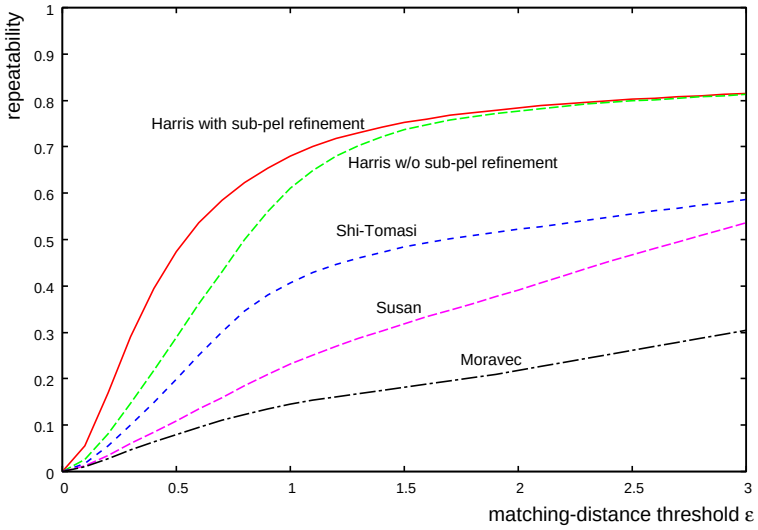
Evaluation results

The diagrams in Figure 3.10 show the *repeatability* for four test sequences[3] and all of the previously described interest-point detectors, using ϵ as a parameter. In each diagram, the repeatability is averaged over the complete sequence length. From these diagrams and the results of other sequences, we see that the Harris detector reaches a *repeatability* between $80 - 90\%$, where saturation is almost reached for $\epsilon \approx 1.5$. Note that a *repeatability* near 100% cannot be reached since new image content appears and old

[3]See Appendix D for a description of the sequence contents.

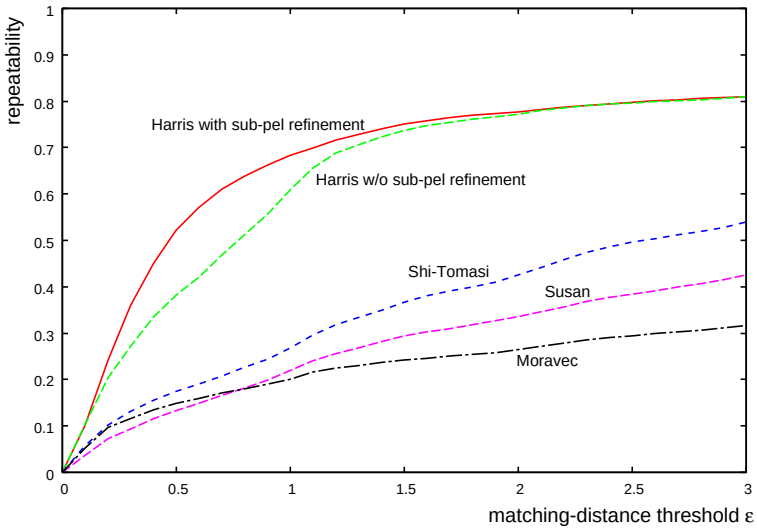


(a) *roma*

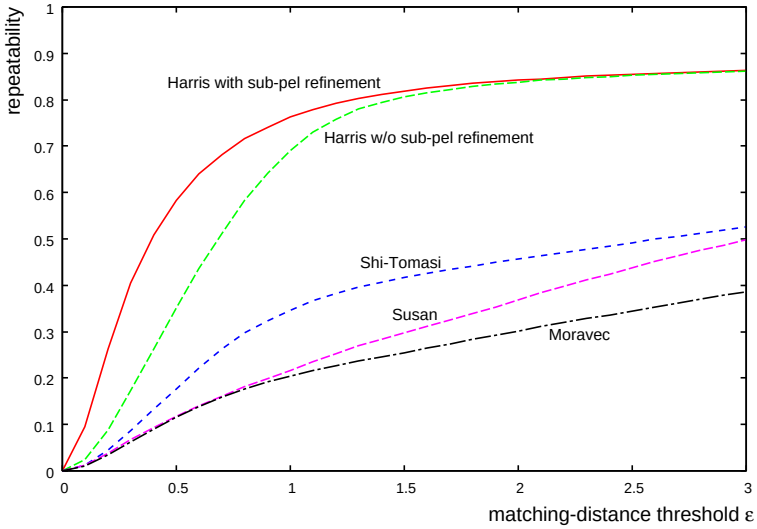


(b) *rail*

Figure 3.10: *Repeatability of the detected feature-points: fraction of corresponding feature-points. Feature-points from one image are transformed into the second image. They match a feature-point in the second image if there is a detected feature-point within a neighborhood ϵ. (Continued in Fig. 3.11.)*



(a) *opera4*



(b) *nature2*

Figure 3.11: *Continuation of Fig. 3.10.*

Accuracy (pixels)	*roma*	*rail*	*opera4*	*nature2*	Average
Harris (sub-pel)	0.34	0.56	0.51	0.46	0.47
Harris (integer)	0.53	0.74	0.65	0.67	0.65
Shi-Tomasi	0.53	0.80	0.95	0.83	0.78
SUSAN	0.75	1.13	0.98	1.11	0.99
Moravec	0.74	1.05	0.76	0.98	0.88

Table 3.1: *Accuracy results for the four test sequences roma, rail, opera4, and nature2.*

content disappears during the sequences. Especially the feature-points near the image border are often impossible to match since their counterpart is outside of the visible image area.

The repeatabilities of all other detectors are clearly below the Harris detector and they also do not reach a clear saturation, which is due to a lower localization accuracy. The Shi-Tomasi detector and SUSAN reach both a repeatability rate between $40 - 70\%$, while the Moravec operator only reaches about $30 - 40\%$.[4]

The results for the *accuracy* of the four sequences are summarized in Table 3.1. It can be noticed that the Harris detector again shows the best performance and that its accuracy can indeed be increased by our sub-pel refinement.

Conclusion

According to our experiments, the Harris detector showed the best performance for repeatability and also for accuracy. While all detectors yield a good accuracy, the repeatability is not satisfactory except for the Harris detector. As a consequence, we selected the Harris detector for our motion-estimation system and we will omit the other detectors in the successive sections. It can also be seen that the sub-pel refinement can in fact increase the accuracy of the Harris detector by about 0.2 pixels.

3.3 Computing feature-correspondences

After feature-points have been extracted for each video frame, it is necessary to establish correspondences between the feature-points. Each correspondence indicates the motion of one image position with a high confidence. Computing the correspondences is subject to a number of problems. As

[4]Since the latter detectors do not show that clear saturation point, we fix ϵ to a defensive value of 2.5 for the successive sections and set $\mathcal{R} = \mathcal{R}_{2.5}$.

we have seen in the previous section, no interest-point detector is able to yield an ideal repeatability. This means that our data-sets will always be contaminated with feature-points that have no matching counterpart in the other image. However, this is not only due to imperfections of the detection algorithms, but it is also caused by the video content itself. One reason are foreground objects that occlude parts that still were visible in the last frame, or objects that uncover previously hidden areas. Comparable to the occlusion problem is the effect that a moving camera results in displacing the video content such that areas at the image border fall outside of the image area while new parts move into the image at the opposite side. Finally, it should be noted that the number of feature-points is usually very high (about 1000-2000 for CIF-resolution video), which emphasizes the need for a computationally efficient algorithm.

The algorithms for feature-correspondences can be coarsely classified into two categories: algorithms that take the texture around a feature-point into account to guide the matching process, and algorithms that only use the geometric point locations. The second type of algorithm has advantages when the transformation between successive images is large, like a rotation of 180°. In this case, it is difficult to define a reliable similarity measure for feature-points. Similarity measures that are invariant to some geometric transformations are a current area of research. However, in our application to video data, the transformation between images is relatively small and we can use a simple measure like the *sum of absolute differences* of the luminance data in the neighborhood to measure feature-point similarity.

3.3.1 Fast greedy algorithm

Several algorithms for computing feature-point correspondences have been proposed in the literature [121, 147]. Some of them are especially designed to allow for large motions [182]. However, for our application, the motion is relatively small and well predictable from previous frames. On the other hand, many feature-points are extracted for typical video content, so that a low computational complexity becomes important. For this reason, we have developed a fast *ad-hoc* algorithm to compute point-feature correspondences. In the following, we first describe the algorithm core, which we subsequently modify to decrease the computational complexity and also enhance the accuracy.

Algorithm core

Let us denote the feature-points in the first image I_t as $\mathbf{p}_i = (x_i, y_i, 1)$ and in the second image I_{t+1} as $\hat{\mathbf{p}}_k = (\hat{x}_k, \hat{y}_k, 1)$. Furthermore, we define a

dissimilarity metric on the pairs of feature-points, using the *sum of absolute errors* distance, defined as

$$m_{i,k} = \sum_{r=-7}^{7} \sum_{s=-7}^{7} \left| I_t(x_i + r, y_i + s) - \hat{I}_{t+1}(\hat{x}_k + r, \hat{y}_k + s) \right| . \qquad (3.19)$$

Finally, we denote the Euclidean distance between the points $\mathbf{p}_i$ and $\hat{\mathbf{p}}_k$ as $d(\mathbf{p}_i, \hat{\mathbf{p}}_k)$. Under the assumption that the motion between successive frames is small, we only consider feature correspondences between points that are within a search-range d_{max}. The purpose of this threshold is to reduce the computational complexity and also to block correspondences between points that are obviously too far away.

The feature-matching algorithm proceeds as follows.

1. We start the algorithm by establishing the set of all admissible candidate correspondences

$$\mathcal{C}_0 = \left\{ \mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_k \;\middle|\; d(\mathbf{p}_i, \hat{\mathbf{p}}_k) \leq d_{max} \right\} . \qquad (3.20)$$

2. For each of the correspondences $\mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_k$ in $\mathcal{C}_0$, we compute and store the dissimilarity measure $m(i,k)$ in a matrix.

3. In a greedy approach, we iteratively select the candidate correspondence $c = \mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_k$ with the lowest dissimilarity measure and add that correspondence to the final set of correspondences: $\mathcal{C} := \mathcal{C} \cup \{c\}$.

4. Since each feature-point may only participate in one correspondence, we remove all candidate correspondences from $\mathcal{C}_0$ that have a feature-point in common with the selected correspondence:

$$\mathcal{C}_0 := \left\{ \mathbf{p}_c \leftrightarrow \hat{\mathbf{p}}_d \;\in \mathcal{C}_0 \;\middle|\; \mathbf{p}_c \neq \mathbf{p}_i \wedge \hat{\mathbf{p}}_d \neq \hat{\mathbf{p}}_k \right\} . \qquad (3.21)$$

The algorithm ends when $\mathcal{C}_0$ is empty or the dissimilarity error exceeds a threshold m_{max}. This threshold m_{max} is required to prevent the algorithm from associating completely dissimilar feature-points which may remain after most correct correspondences have been found.

Reducing the search-range with motion prediction

The computational complexity of the above algorithm is primarily influenced by the size of the initial candidate set $\mathcal{C}_0$, since for each candidate correspondence, the dissimilarity measure has to be calculated. One possible approach to reduce the candidate set would be to reduce the search-range

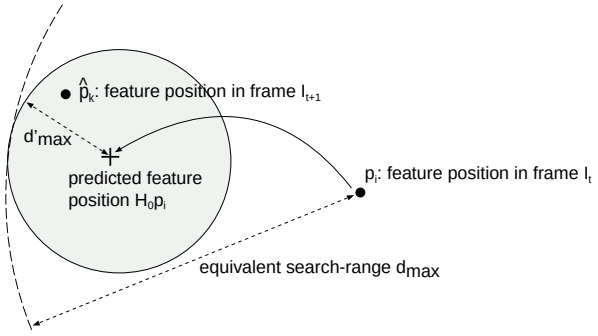


Figure 3.12: *Reducing the required search-range with a motion prediction. Assuming that the past motion* $\mathbf{H}_0$ *does not change much, the feature-point* $\mathbf{p}_i$ *is projected into the future to the position* $\mathbf{H}_0\mathbf{p}_i$*. This allows for a smaller search-range* d'_{max} *compared to the search-range* d_{max} *in the case without prediction.*

d_{max}, but this would make the above algorithm unusable for sequences with fast motion. Using a similar prediction approach as has been used previously for block-based motion [35, 36], we assume that the motion between successive frames is smooth without abrupt changes. This allows us to use the motion model from the previous pair of frames as a predictor to estimate the position of the feature-points in the current frame. Hence, we can limit the search-range for matching features to only a small neighborhood around this estimated feature position. Using the motion model that we computed for the previous pair of images, we can replace Eq. (3.20) by

$$\mathcal{C}_0 = \left\{ \mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_k \;\middle|\; d(\mathbf{H}_0\mathbf{p}_i, \hat{\mathbf{p}}_k) \leq d'_{max} \right\}, \tag{3.22}$$

where $\mathbf{H}_0$ is the motion model prediction and d'_{max} is the radius of the search-range around the predicted position (Fig. 3.12). Because the predicted feature position will be closer to the actual position, we can choose d'_{max} smaller than d_{max}, which reduces the computational complexity without sacrifying the ability to handle large motions. The predicted displacement is not limited, so that the actual effective search-range is unlimited. However, we have the restriction that the motion may not change in speed too quickly.

Fast neighborhood search with 2-D bucket sort

The evaluation of Eq. (3.20) and (3.22) requires finding the set of feature-points $\hat{\mathbf{p}}_k$, that are within a maximum distance d_{max} around a position $\mathbf{H}_0\mathbf{p}_i$. The naive approach to iterate through all feature-points and com-

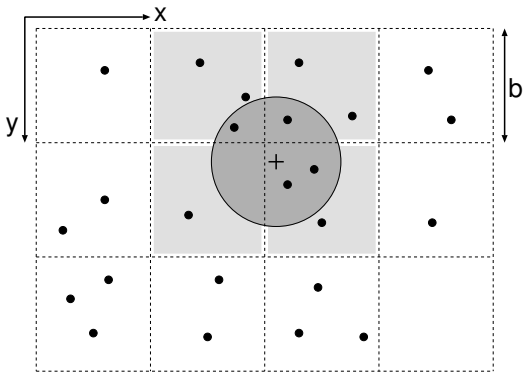


Figure 3.13: *The image area is divided into small squares (buckets) of size $b \times b$. A list of feature-points is associated with each bucket. To get a list of features in a neighborhood of a given point, only the features in the buckets intersecting the neighborhood have to be checked (indicated with grey).*

pute the distance requires substantial computation time because the number of feature-points can be larger than 1000. Consequently, 1000×1000 comparisons would be required for each frame. We can reduce this considerably by storing the feature-points in a data structure which allows for an efficient search for close points. A simple solution is to partition the image area into a lattice of small square cells where each cell (m, n) consists of a set $\mathcal{B}_{m,n}$, storing the feature-points located in the area of this cell:

$$\mathcal{B}_{m,n} = \left\{ \hat{\mathbf{p}}_k = (x, y) \;\middle|\; \left\lfloor \frac{x}{b} \right\rfloor = m \wedge \left\lfloor \frac{y}{b} \right\rfloor = n \right\}. \tag{3.23}$$

The parameter b is the cell width and can be chosen as, e.g., $b \approx d'_{max}$, but the actual choice is not critical. Locating the set of features within a maximum distance can now be carried out efficiently by first determining which cells intersect the search-range (Fig. 3.13). The points within cells that are completely covered by the search-range can be taken into the solution set without further testing. Points within cells that are only partially covered by the search-range have to be tested individually.

3.3.2 Evaluation

A typical output of the algorithm is visualized in Figure 3.14. The correspondences are drawn with lines where the small circle indicates the feature position in the second frame. Unmatched feature-points are drawn with

larger circles (unfilled for features of the first frame and filled for features of the second frame). Most unmatched features are located along the image border, since this part of the image is visible in only one of the two frames. Other sporadic unmatched features are due to a non-perfect repeatability in the interest-point detector. It is obvious that the number of unmatched features increases quickly if the search-range is too small and motion prediction is not used (see Figure 3.15). However, even in this case, a correct estimate of the background motion would be possible, because enough correspondences have still been found.

In general, a large search-range is required if the motion is very fast. But unfortunately, a large search-range increases the probability to find non-correct matches (see Fig. 3.16(a)), and it also increases the computation time. However, if we use motion prediction to relocate the center of the search-range, we can obtain the correct correspondences with a much smaller search-range (see Fig. 3.16(b)). At the same time, the number of wrong matches is reduced, because the algorithm locks to the camera motion $\mathbf{H}_0$ and since it only detects correspondences that fit into the global-motion model.

The effect of motion prediction is evaluated further in Figure 3.17 for four test sequences. Depicted is the recall rate $|\mathcal{C} \cap \mathcal{R}|/|\mathcal{R}|$ (the fraction of correctly found correspondences) depending on the search-range. Also shown is the fraction of the total correspondences found $|\mathcal{C}|/|\mathcal{R}|$ (including the wrong matches). All values are averaged over the complete sequence length. The diagrams show clearly that a much smaller search-range of only about two pixels (on the average) is required when motion prediction is enabled, whereas a much larger range is required without prediction. Obviously, both approaches converge for larger search-ranges.

In a second experiment, we explored the dependency of the recall rate on the maximum matching error that is allowed for a feature-correspondence. Since the image motion in general is non-translatorial, but the feature similarity is computed assuming only a translatorial model, the feature error increases for larger non-translatorial transforms. Hence, it is clear that a higher threshold on the matching error allows more feature-correspondences to be established. On the other hand, a higher threshold also induces more erroneous features matches. Figure 3.19 again shows the recall rate, but now for different maximum matching errors. Since we found that for higher thresholds, the number of correct correspondences as well as the number of errors increases, a good trade-off value has to be found. Based on the results using our test sequences, we chose $m_{max} = 5000$ as threshold. However, we also noticed that the choice of this threshold is non-critical, as the final result of the motion estimation was similar over a wide range of m_{max}.

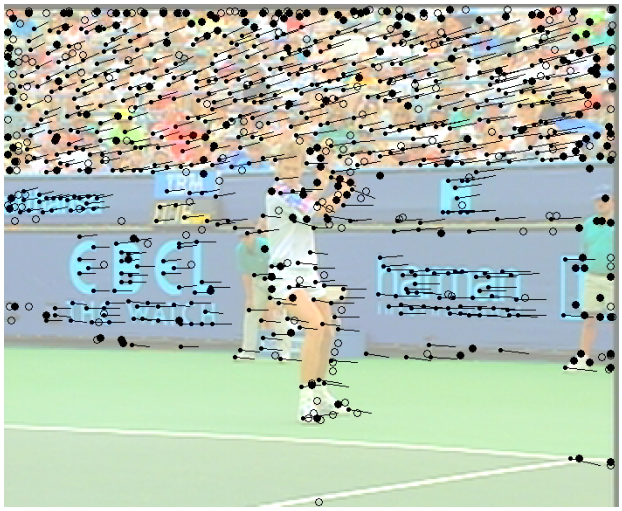


Figure 3.14: *Typical output of our greedy point-correspondence algorithm. Spurious unmatched feature-points exist from the imperfect interest-point detector. Since these points are not passed to the next stage, they do not degrade the total algorithm output.*

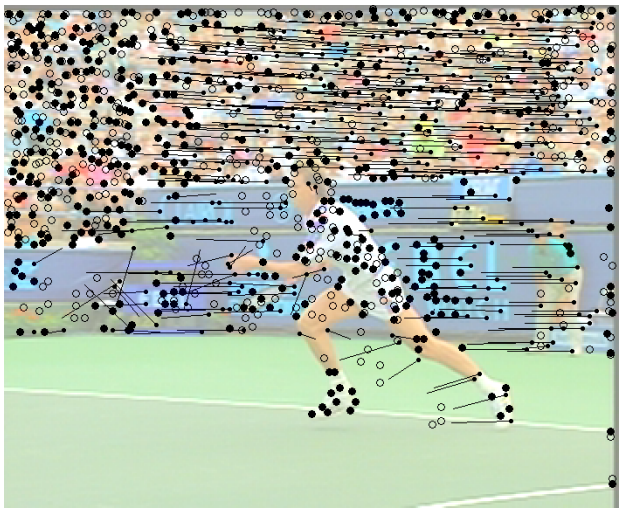
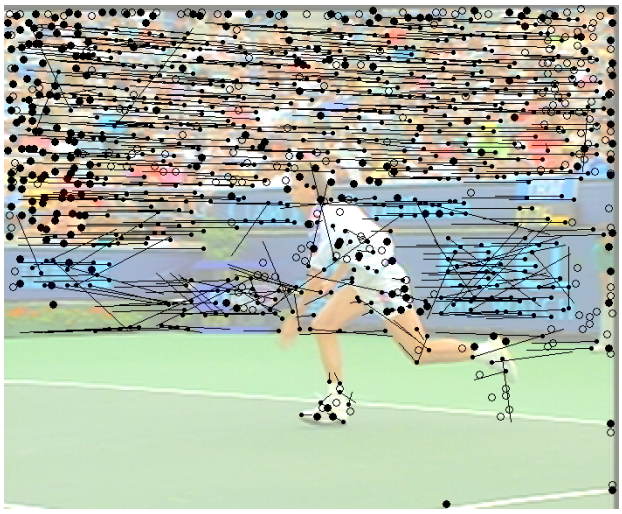
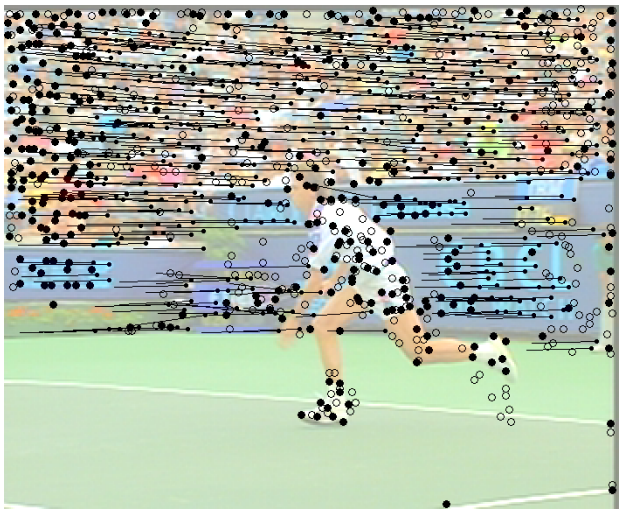


Figure 3.15: *Computed motion between frames 236 and 238 without motion prediction. It can be seen that the search-range (40 pixels) is not sufficient to find correspondences at the left part of the audience and in the tennis-player object. (Motion on the left side is faster since a left pan is combined with a zoom-out.)*

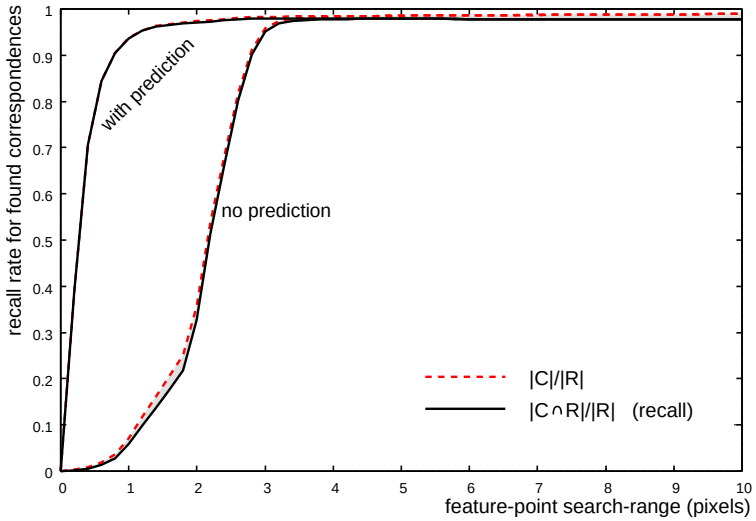


(a) No motion prediction. A large search-range (70 pixels) is required, which also increases the number of wrong matches.

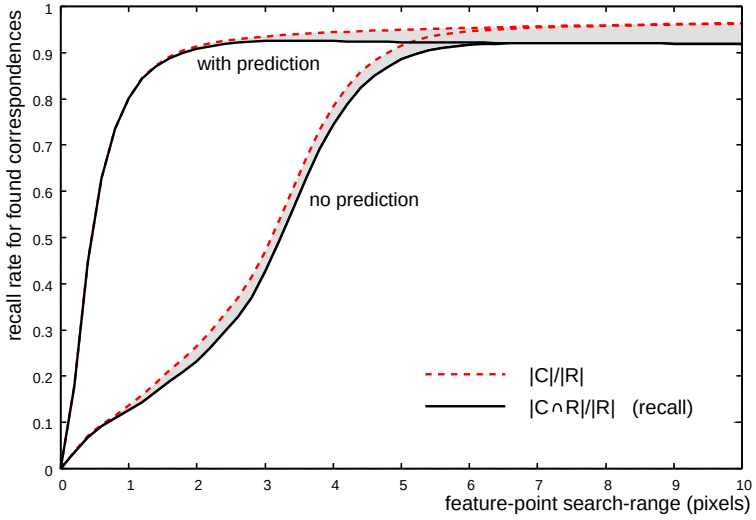


(b) Enabled motion prediction with small search-range (8 pixels). The algorithm locks to the camera motion and the number of wrong matches is reduced considerably.

Figure 3.16: *Comparison of the correpondence algorithm without motion prediction and with motion prediction for a scene with fast motion.*

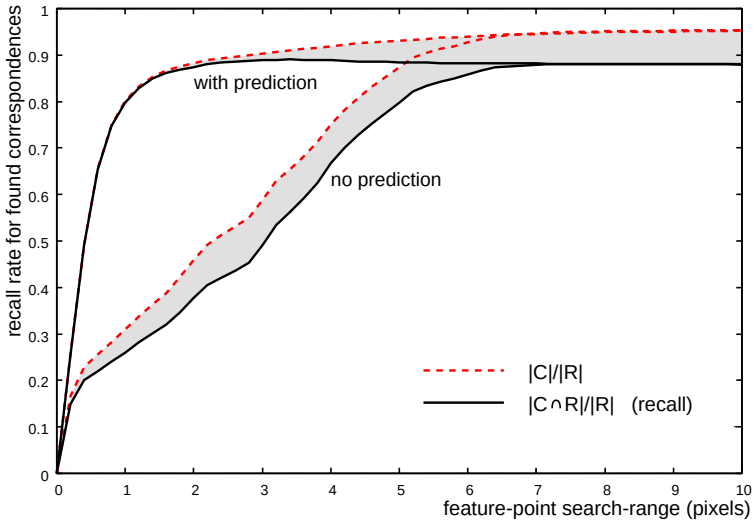


(a) *roma*

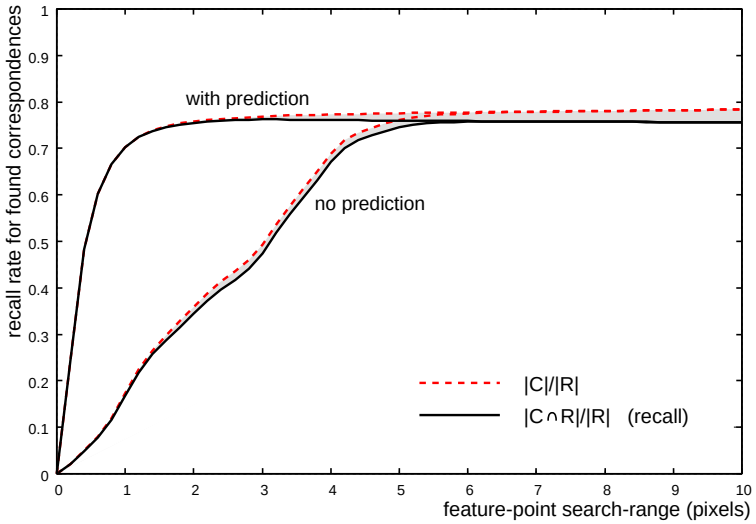


(b) *rail*

Figure 3.17: *Recall rate of correct correspondences for different search-ranges d_{max} (solid line). The maximum matching error m_{max} is fixed to 5000. For comparison, the total number of returned correspondences (also normalized to $|\mathcal{R}|$) is shown with a dashed line. The wrong correspondences can be seen as the difference between the two values. (Continued in Fig. 3.18.)*

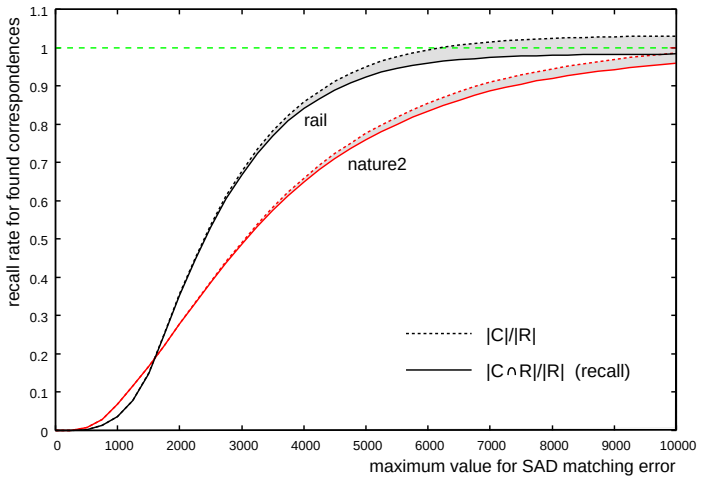


(a) *opera4*

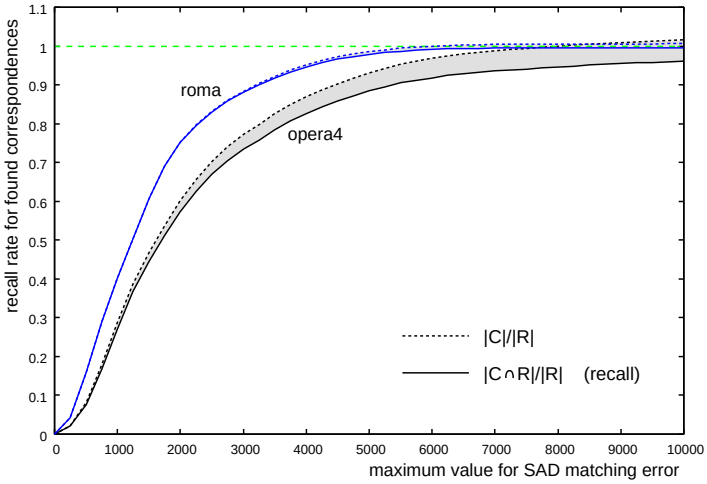


(b) *nature2*

Figure 3.18: *Continuation of Fig. 3.17.*



(a)



(b)

Figure 3.19: *Recall rate of correct correspondences for different maximum matching errors m_{max} (solid line). The search-range is fixed to $d'_{max} = 3$, motion prediction is turned on. For comparison, the total number of returned correspondences (also normalized to $|\mathcal{R}|$) is shown with a dashed line. The wrong correspondences can be seen as the difference between the two values.*

3.4 Summary

This chapter presented the feature-point detection and matching steps of the feature-based motion estimator. First, we evaluated four different interest-point detectors for their performance in our application. In particular, we defined and measured the *repeatability* and the *accuracy* of the detectors. The repeatability is the number of features that are detected in both images of a pair. A high repeatability is desired, since only features that are detected in both images can establish a correct correspondence for the successive motion estimation step. The accuracy of an interest-point detector is the variance of the detected feature position in different images. A high accuracy is desired, because inconsistent placement of the feature points would finally lead to less accurate motion parameters. Based on our experiments, we concluded that the Harris interest-point detector with our sub-pel refinement provided the best repeatability of approx. 90% and also the highest accuracy of about 0.5 pixels.

Furthermore, we presented the feature-matching algorithm. The algorithm is a greedy highest-confidence-first algorithm that first groups features for which the SAD matching error of a small window around the feature is smallest. We added a motion prediction step that predicts the position of a feature in the new frame based on the motion model between the last pair of frames. The feature matching is limited to a small neighborhood around this predicted position. This motion prediction has two advantages: first, the small neighborhood allows for a computationally fast search, compared to considering all detected features. Second, it prevents that features are matched that deviate much from the current camera motion. This effectively decreases the number of wrong matches.

Architecture of the feature-point detection and matching

The data-flow of the feature-point detection and the feature matching step is depicted in Figure 3.20. For a pair of temporally successive pictures, feature-points are extracted with the Harris detector. Note that for each step only one input frame has to be processed, since the features of the other frame can be reused from the last step. The features $\mathcal{F}_t$ are projected onto the next frame, using the previously computed camera-motion model $\mathbf{H}_{t;t-1}$. Matching the two feature sets $\mathcal{F}_t$ and $\mathcal{F}_{t+1}$ results in a set of correspondences $\{\mathbf{p}_i \leftrightarrow \hat{\mathbf{p}}_i\}$ that will be used to determine the camera-motion parameters (as described in the subsequent chapter). The computed camera-motion parameters are then used to predict the feature positions in the next pair of frames.

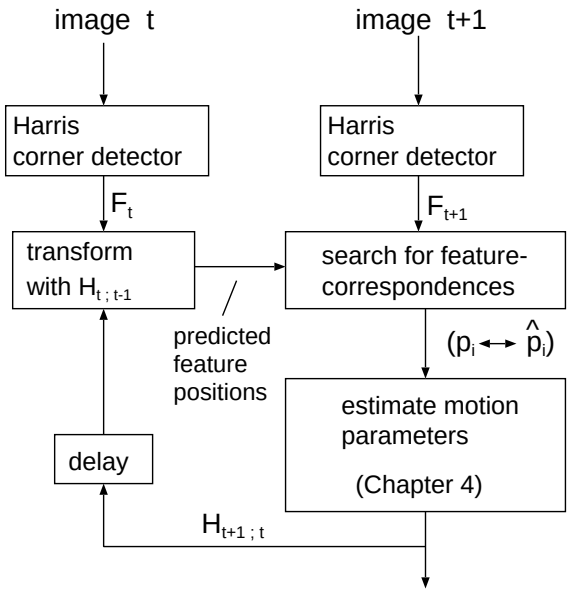


Figure 3.20: *Feature-point detection and matching of corresponding features.*