CHAPTER 22

DIGITAL VIDEO PROCESSING TECHNIQUES AND APPLICATIONS

WHAT WILL WE LEARN?

- What is motion estimation (ME) and why is it relevant?
- Which techniques and algorithms can be used to estimate motion within a video sequence?
- Which techniques can be used to filter a video sequence?
- What is the role of motion compensation (MC) in video filtering?

22.1 FUNDAMENTALS OF MOTION ESTIMATION AND MOTION COMPENSATION

Motion is an essential aspect of video sequences. The ability to estimate, analyze, and compensate for relative motion is a common requirement of many video processing, analysis and compression algorithms and techniques. In this section, we present the fundamental concepts associated with motion estimation and compensation.

Motion estimation is the process of analyzing successive frames in a video sequence to identify objects that are in motion. The motion of an object is usually described by a two-dimensional (2D) *motion vector* (MV), which encodes the length and direction of motion.

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

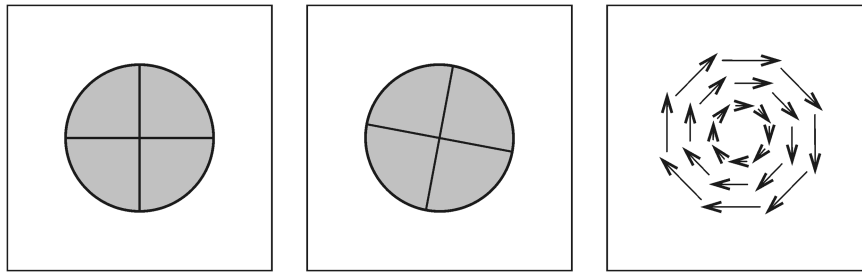

FIGURE 22.1 Two frames at different time instants (t_1 and $t_1 + \Delta_t$) and the resulting optical flow. Redrawn from [SHB08].

Many two-dimensional motion estimation algorithms are based on the concept of *optical flow*, which can be defined as the pattern of *apparent* motion of objects, surfaces, and edges in a visual scene. The optical flow—and consequently the apparent motion—can be computed by measuring variations (i.e., gradients) of intensity values over time. The resulting pattern is a vector field—known as the *optical flow field*—containing the motion vectors for each support region (e.g., pixel, block, or region) in the frame, representing the best approximation of the 3D motion of object points across the 2D frame. Figure 22.1 shows a schematic example of two consecutive frames and the resulting optical flow.

Optical flow computations[1] are based on two assumptions [SHB08]:

1. The observed brightness of an object is constant over time. This is referred to as the *constant intensity assumption*.
2. Points that are spatially close to one another in the image plane tend to move in a similar manner (the *velocity smoothness* constraint).

The two-dimensional motion estimation techniques aim at encoding motion information from intensity and color values obtained from the frames in a video sequence. This is a complex problem for many reasons, such as the following:

- The actual scene is three dimensional (and so are the objects in it), but 2D motion estimation algorithms have access only to a 2D representation of it. Consequently, these algorithms must rely on the *apparent motion* of the objects relative to the frame, which does not always correspond to true motion in the original scene.

  There are many cases in which actual object motion would go undetected or—conversely—a 2D motion estimation algorithm would estimate motion where none existed. An example of the former would be a sphere rotating on its own

[1]The complete mathematical formulation of optical flow methods is beyond the scope of this text. Refer to "Learn More About It" section at the end of the chapter for references.

axis. The latter problem can arise as a result of lighting variations, distracting objects, or many other scenarios.

- We can speak of motion only in *relative* terms, that is, by measuring how camera (or observer), objects of interest, and background move relatively to one another. There are six general cases to consider:
  1. Still camera, single moving object, constant background.
  2. Still camera, several moving objects, constant background.
  3. Still camera, single moving object, moving (cluttered) background.
  4. Still camera, several moving objects, moving (cluttered) background.
  5. Moving camera, fixed object(s), constant background.
  6. Moving camera, several moving objects, moving (cluttered) background.

  Most motion estimation algorithms in use today can handle the first two cases rather easily. There is a significant amount of ongoing work on cases where the background is complex and contains motion (e.g., wavering tree branches, light reflected on the ocean, etc.). The fifth case is usually treated as a separate problem, for which the goal is to determine the type and amount of camera movement. Finally, the sixth category is beyond the reach of contemporary computer vision solutions.
- There is an inherent ambiguity in the process of estimating motion based on edges of objects within the frame, which is known in the vision literature as the *aperture problem*, illustrated in Figure 22.2. If we view the image through a small circular aperture, we cannot tell exactly where a point (x_i, y_i) on an edge moves to. Optical flow-based ME algorithms usually rely on the motion vector perpendicular to the edge (indicated as $\vec{u}_n$), suggesting that the edge has moved up and to the right in this case. Clearly, it is possible that the actual motion was

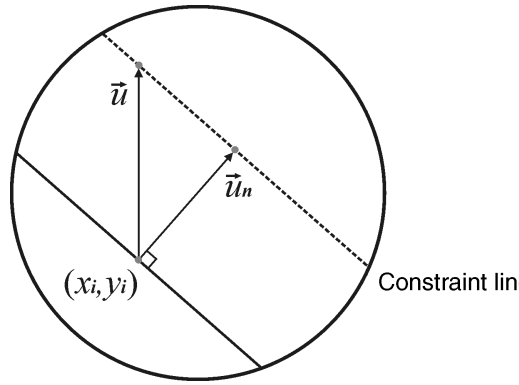

FIGURE 22.2 The aperture problem.

different—from purely up (indicated as $\vec{u}$) to purely to the right—but there is no way to know it for sure.

- ME algorithms must work at a level of granularity that is appropriate to the task at hand. Representing a MV for each pixel is usually not a good idea, and neither is using a single MV for the entire frame. Many algorithms find a compromise between these two extreme cases, by working with nonoverlapping blocks.
- In many cases (e.g., object tracking applications), we might be interested in computing the motion of *objects* in the scene, which presupposes that they can be accessed as individual entities. This increases the complexity of the overall solution, since it would require (accurate) object segmentation *before* the ME calculations.
- ME methods strongly depend on the intended application. For example, in video compression applications, the main goal of ME algorithms is to generate motion vectors that allow a decoder to re-create a frame based on the motion-compensated differences between the current frame and a previously decoded frame, while saving bits in the process. In motion-compensated standards conversion, on the other hand, the goal is to align the interpolation axis to the direction of motion of objects across multiple frames, and therefore reduce artifacts such as *judder*.

Motion compensation is the process by which the results of the motion estimation algorithms (usually encoded in the form of motion vectors) are employed to improve the performance of video processing algorithms. In its simplest case (e.g., a linear camera motion such as *panning*), motion compensation computes where an object of interest will be in an intermediate target field/frame and then shifts the object to that position in each of the source fields/frames.

In more general cases, the results of the motion estimation algorithm (i.e., the motion vectors) are used to create the equivalent of interpolation axes, aligned with the trajectory of each moving object (Figure 22.3). This causes a significant reduction in motion and temporal aliasing and has a dramatic positive effect on the output of motion-compensated algorithms for tasks such as interframe filtering (see Section 22.4.2), de-interlacing (see Section 21.3.1), and standards conversion (see Section 21.3), among many others. The overall performance of the motion-compensated video processing algorithms is determined primarily by the accuracy of the motion vectors, which underscores the importance of the motion estimation stage, which is described in more detail in the following sections.

22.2 GENERAL METHODOLOGIES IN MOTION ESTIMATION

The 2D motion estimation techniques described in this chapter address the problem of estimating the motion between two frames: $f(x, y, t_1)$, which will be called the *anchor frame*, and $f(x, y, t_2)$, which will be referred to as the *target frame*. The anchor frame may be either before or after the target frame in time. We will call it *forward*

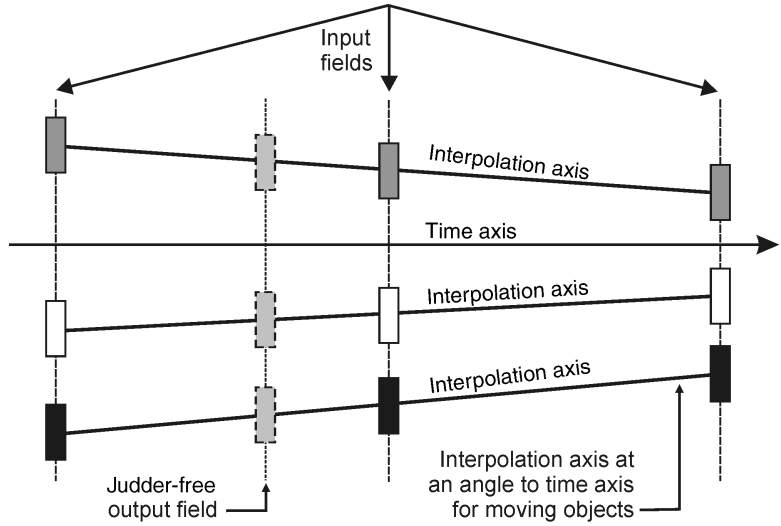

FIGURE 22.3 Motion compensation: interpolation axes are aligned with each moving object. Redrawn from [Wat94b].

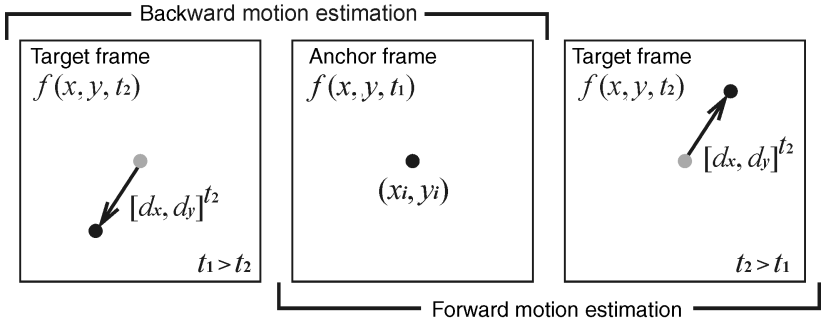

FIGURE 22.4 Anchor and target frames in forward and backward motion estimation.

motion estimation the case where $t_2 > t_1$ and *backward motion estimation* the case where $t_1 > t_2$. A motion vector $\mathbf{d}(\mathbf{x}) = [d_x d_y]^T$ will represent the displacement of a particular point $\mathbf{x} = (x_i, y_i)$ between time t_1 and t_2. Figure 22.4 illustrates these concepts and associated notation.

There are two main categories of ME approaches:

- *Feature-Based Methods*: Establish a correspondence between pairs of selected feature points in the two frames and attempt to fit them into a motion model using methods such as least-squares fitting. Feature-based methods are more often used in applications such as object tracking and 3D reconstruction from 2D video.

- *Intensity-Based Methods*: Rely on the constant intensity assumption described earlier and approach the motion estimation problem from the perspective of an optimization problem, for which three key questions must be answered:
  - How to represent the motion field?
  - Which criteria to use to estimate motion parameters?
  - How to search for optimal motion parameters?

  This section focuses exclusively on intensity-based methods. The answers to the three questions above will appear in Sections 22.2.1–22.2.3, respectively.

22.2.1 Motion Representation

Motion representation methods vary in terms of the size (and shape) of the region of support, which can vary among the following four types (illustrated in Figure 22.5):

1. *Global*: In this case, the region of support is the entire frame and a single motion model applies to all points within the frame. This is the most constrained motion model, but it is also the one that requires fewest parameters to estimate. Global motion representation is often used to estimate camera-induced motion, that is, apparent motion caused by camera operations such as pan, zoom, or tilt.

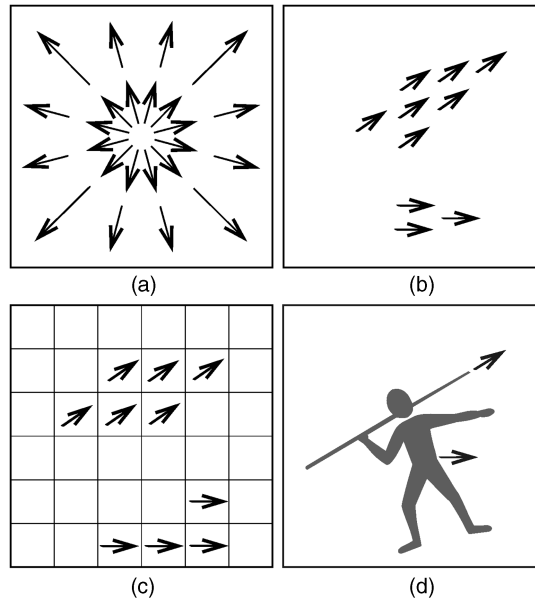

FIGURE 22.5 Motion estimation methods: (a) global; (b) pixel-based; (c) block-based; (d) object-based.

2. *Pixel-Based*: This model produces the densest possible motion field, with one MV per pixel, representing the 2D displacement of that point across successive frames. A smoothness constraint between adjacent MVs is often employed to ensure a more coherent motion representation. The computational complexity of pixel-based methods is high, due to the enormous number of values (twice the number of pixels in the frame) that need to be estimated.
3. *Block-Based*: In this case, the entire frame is divided into nonoverlapping blocks, and the problem is reduced to finding the MV that best represent the motion associated with each block. This model is widely used in video compression standards such as H.261, H.263, MPEG-1, and MPEG-2.
4. *Region-Based*: In this model, the entire frame is divided into irregularly shaped regions, with each region corresponding to an object or subobject with consistent motion, which can be represented by a few parameters. It is used in the MPEG-4 standard.

22.2.2 Motion Estimation Criteria

Once a model and region of support have been chosen, the problem shifts to the estimation of the model parameters. One of the most popular motion estimation criteria—and the only one described in this book—is based on the displaced frame difference (DFD). It consists in computing the differences in intensity between every point in the anchor frame and the corresponding point in the target frame. Mathematically, the objective function can be written as

$$E_{\mathrm{DFD}}(\mathbf{d}) = \sum_{\mathbf{x}\in\mathcal{R}} (f_2(\mathbf{x}+\mathbf{d}) - f_1(\mathbf{x}))^p \tag{22.1}$$

where $f_1(\mathbf{x})$ is the pixel value at $\mathbf{x}$ in time t_1, $f_2(\mathbf{x}+d)$ is the pixel value at $(\mathbf{x}+\mathbf{d})$ in time t_2, $\mathbf{d}$ is the estimated displacement, and $\mathcal{R}$ is the region of support.

When $p = 1$, the error $E_{\mathrm{DFD}}(\mathbf{d})$ is called the *mean absolute difference* (MAD), and when $p = 2$, it is called the *mean squared error* (MSE).

22.2.3 Optimization Methods

The error functions described by equation (22.1) can be minimized using different methods. The three most common optimization methods are as follows:

- *Exhaustive Search*: As the name suggests, this method searches through all possible parameter value combinations and is guaranteed to reach the global optimal at the expense of significant computational cost. It is typically used in conjunction with the MAD criterion described earlier.
- *Gradient-Based Search*: Uses numerical methods for gradient calculation and reaches the local optimal point closest to the initial solution. It is typically used in conjunction with the MSE criterion described earlier.

- *Multiresolution Search*: This method searches the motion parameters starting at a coarse resolution and propagating the partial solutions to finer resolutions, which can then be successively refined. It is a good trade-off between speed (i.e., it is faster than exhaustive search) and does not suffer from the problem of being trapped into a local minimum (as the gradient-based method does).

22.3 MOTION ESTIMATION ALGORITHMS

In this section, we present a selected subset of block-based ME algorithms. These algorithms assume that a frame has been divided into M nonoverlapping blocks that together cover the entire frame. Moreover, the motion in each block is assumed to be constant, that is, it is assumed that the entire block undergoes a translation that can be encoded in the associated motion vector. The problem of block-based ME algorithms is to find the best MV for each block. These algorithms are also called *block matching algorithms* (BMA).

22.3.1 Exhaustive Search Block Matching Algorithm

The exhaustive search block matching algorithm (EBMA) (Figure 22.6) works as follows: for each block $\mathcal{B}_{\mathrm{m}}$ in the anchor frame, it searches all candidate blocks (within a certain range) $\mathcal{B}'_{\mathrm{m}}$ in the target frame, and selects the one with the minimum error, that is, the one that optimizes a cost function such as MAD. The search range is indicated in Figure 22.6 by R_x and R_y.

The estimation accuracy of the EBMA method is determined by the step size for the block displacements. In the simplest case, the step size is one pixel, and the process is known as *integer-pixel accuracy search*. For more accurate motion

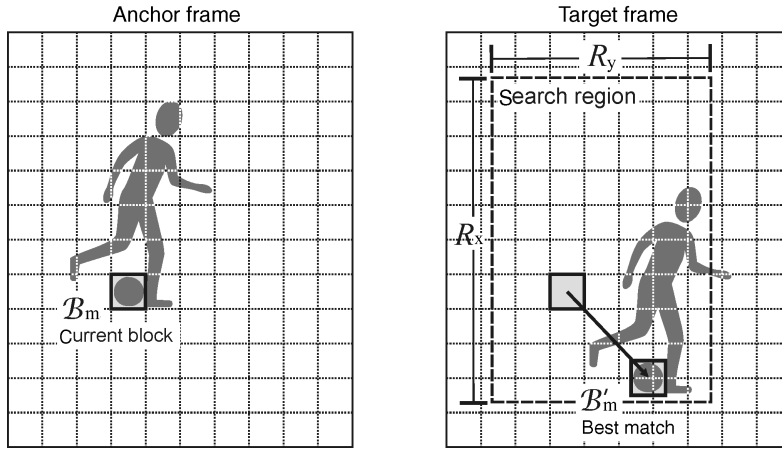

FIGURE 22.6 Exhaustive search block matching algorithm (EBMA).

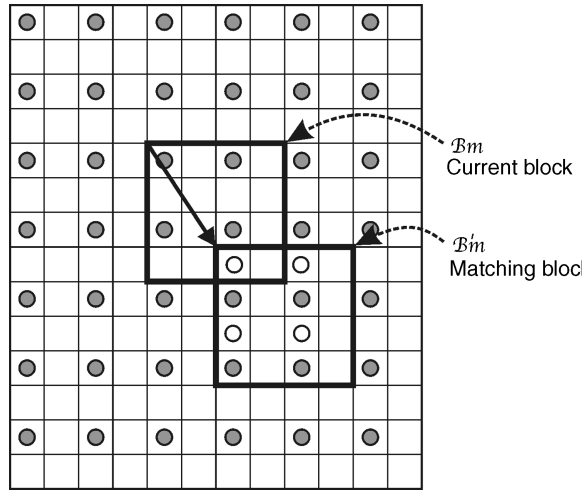

FIGURE 22.7 Block matching with half-pixel accuracy. The MV in this case is (1, 1.5). Redrawn from [WOZ02].

representation, *half-pixel accuracy search* is often used. In order to achieve subpixel accuracy, the target frame must be up-sampled (with the proper interpolation method, usually bilinear interpolation) by the proper factor. Figure 22.7 illustrates the process of block matching using half-pixel accuracy. In this figure, solid circles correspond to samples that exist in the original target frame, whereas hollow circles correspond to interpolated samples. In Tutorial 22.1 (page 579), you will learn how to perform motion estimation using the EBMA method (using both full-pixel and half-pixel accuracy) in MATLAB.

The EBMA algorithm has several weaknesses:

- It causes a blocking artifact (noticeable discontinuity across block boundaries) in the predicted frame.
- The resulting motion field can be somewhat chaotic because MVs are independently estimated from block to block.
- Flat regions are more likely to produce wrong MVs because motion is indeterminate when spatial gradient is near zero (a problem shared by all optical flow-based methods).

■ EXAMPLE 22.1

Figure 22.8 shows the full-pixel EBMA algorithm at work. Parts (a) and (b) show the target and anchor frame, respectively. Part (c) shows the motion field, which exemplifies the last two limitations described earlier. Finally, part (d) shows a reconstructed frame, that is, a frame obtained by computing the estimated motion applied to the target frame. Ideally, part (d) should result in a frame identical to the one in

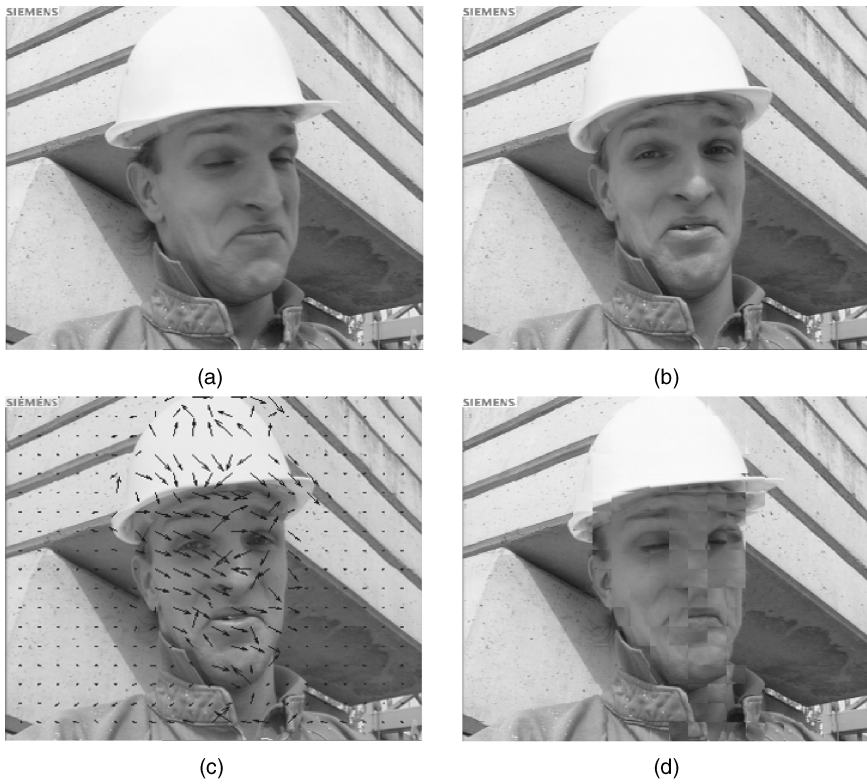

FIGURE 22.8 EBMA example: (a) target frame; (b) anchor frame; (c) motion field overlapped on anchor frame; (d) reconstructed frame.

part (b), but this is clearly not the case: the presence of blocking artifacts (besides other imperfections) in the reconstructed frame is clearly visible.

22.3.2 Fast Algorithms

The EBMA algorithm is computationally expensive (and significantly more so if fractional pixel accuracy is used), which motivated the development of several fast algorithms for block matching. The two most popular fast search algorithms are the 2D log and the three-step search methods. They are briefly described next.

2D Log Search Method The 2D log search method (Figure 22.9) starts from the position corresponding to zero displacement and tests five search points, arranged in a diamond shape. It selects the block that yields minimum error (usually MAD is the chosen figure of merit) and uses it to form a new search region around it. If the best matching point is the center point, it proceeds with a new search region that has an offset of half the amount of the previous offset; otherwise, the offset remains

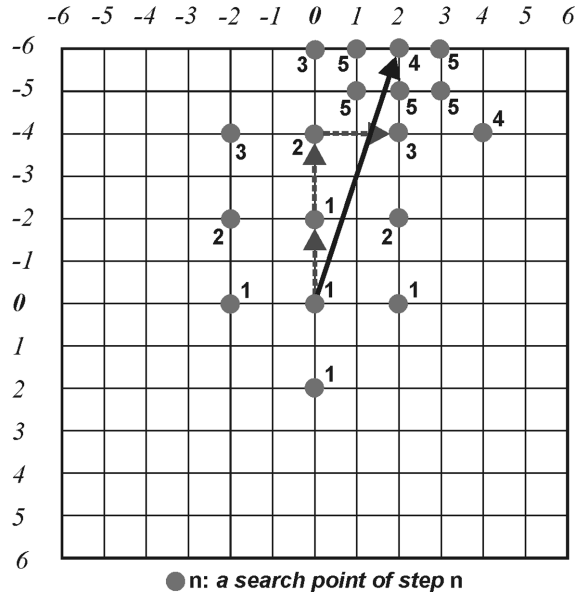

FIGURE 22.9 2D log search. In this case, the final MV is (−6, 2). Redrawn from [MPG85].

the same. The process is repeated for successively smaller ranges, until the offset is equal to 1.

Three-Step Search Method The three-step search method (Figure 22.10) starts from the position corresponding to zero displacement and tests nine search points, with offset $p/2$, where p is half of the search range in each direction (horizontal or vertical). It selects the block that yields minimum error and uses it to form a new search region around it. The new search region has an offset of half the amount of the previous offset. The process is repeated for successively smaller ranges, until the offset is equal to 1.

22.3.3 Hierarchical Block Matching Algorithm

The hierarchical block matching algorithm (HBMA) is a multiresolution ME algorithm. It works by first estimating the motion vectors in a coarse resolution using a low-pass filtered, down-sampled frame pair. It then proceeds to modify and refine the initial solution using successively finer resolutions and proportionally smaller search ranges. The most common approach uses a pyramid structure (Figure 22.11) in which the resolution is reduced by half in both dimensions between successive levels. The HBMA algorithm strikes a compromise between execution time and quality of results: it has lower computational cost than the EBMA counterpart and produces better quality motion vectors than the fast algorithms described earlier. In Tutorial 22.1

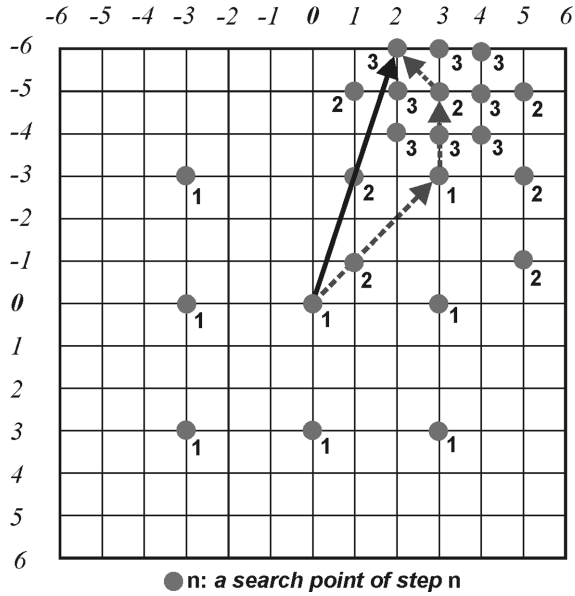

FIGURE 22.10 Three-step search. In this case, the final MV is (−6, 2). Redrawn from [MPG85].

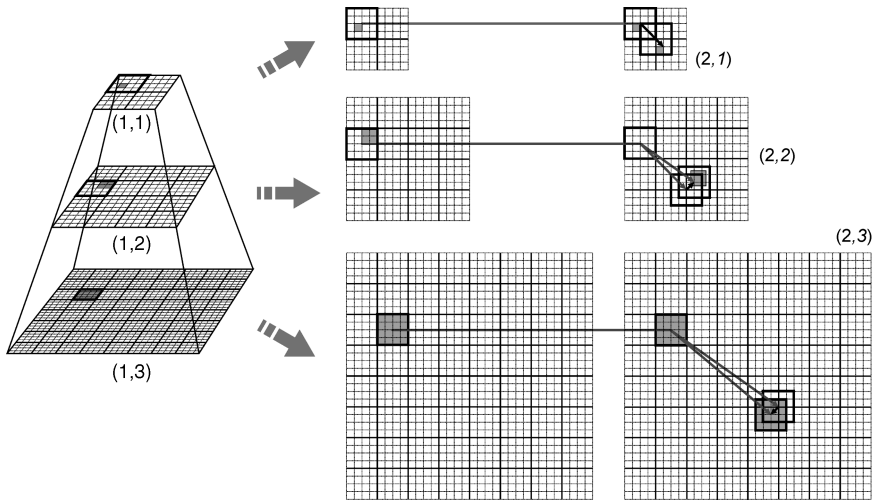

FIGURE 22.11 Hierarchical block matching algorithm (HBMA) using three levels. Redrawn from [WOZ02].

(page 579), you will learn how to perform motion estimation using the HBMA method in MATLAB.

22.3.4 Phase Correlation Method

The phase correlation method is a frequency-domain motion estimation technique that uses a normalized cross-correlation function computed in the frequency domain to estimate the relative shift between two blocks (one in the anchor frame, another in the target frame). The basic idea consists in performing spectral analysis on two successive fields and computing the phase differences. These differences undergo a reverse transform, which reveals peaks whose positions correspond to motions between fields.

1. Compute the discrete Fourier transform (DFT) of each block in the target and anchor frames (which we will denote frames k and $k + 1$).
2. Calculate the normalized cross-power spectrum between the resulting DFTs according to equation (22.2):

$$C_{k,k+1} = \frac{\mathfrak{F}_{k+1}\mathfrak{F}_k^*}{|\mathfrak{F}_{k+1}\mathfrak{F}_k^*|} \tag{22.2}$$

   where $\mathfrak{F}_k$ and $\mathfrak{F}_{k+1}$ are the DFT of two blocks at the same spatial location in frames k and $k + 1$ and * denotes the complex conjugate.
3. Compute the *phase correlation function*, PCF($\mathbf{x}$), defined as the inverse DFT of $C_{k,k+1}$, as follows:

$$\text{PCF}(\mathbf{x}) = \mathfrak{F}^{-1}(C_{k,k+1}) = \delta(\mathbf{x} + \mathbf{d}) \tag{22.3}$$

   where $\mathbf{d}$ is the displacement vector.
4. Locate the peaks of the PCF.

The basic algorithm described above, however, has a major limitation. It lacks the ability to tell which peak corresponds to which moving object in the frames. This problem can be resolved by adding a (selective) block matching stage, used to distinguish between valid motion vectors and false alarms. The modified phase correlation method has the following strengths: it is fast, subpixel accurate, and amplitude (brightness) independent. In Tutorial 22.1 (page 579), you will learn how to perform motion estimation using the phase correlation method in MATLAB.

22.4 VIDEO ENHANCEMENT AND NOISE REDUCTION

In this section, we provide a brief overview of how video sequences can be processed using filtering techniques aimed at enhancing the image quality or reducing noise and other artifacts. The simplest approach consists of applying (spatial- or

frequency-domain) image processing algorithms to each individual frame: this will be called *intraframe filtering*. Many techniques described in Part I can be used to implement intraframe filtering, such as averaging filter and its variants, median and weighted median filters, or Wiener filters, to mention but a few.

Better results, however, can be obtained by exploiting the contents of adjacent frames with significant redundant information, using video-specific (spatiotemporal) algorithms, which will be referred to as *interframe filtering* (Section 22.4.2). In Tutorial 22.2 (page 585), you will experiment with inter- and intraframe video filtering techniques in MATLAB.

22.4.1 Noise Reduction in Video

Video acquisition, transmission, and recording systems are not perfect. Consequently, video sequences may be subject to different types of noise and distortion.

For analog video, the most common types are as follows:

- *Thermal (snow) Noise*: Caused by temperature-dependent random signal-level fluctuations in amplifiers or signal attenuation due to rain. The visible effect on the screen resembles a snowstorm.
- *Composite Triple Beats (CTB)*: Caused by cumulative effect of third-order intermodulation products and overdriven amplifiers. Produces graininess and tearing in the horizontal lines of the screen.
- *Composite Second-Order Beat (CSO)*: Caused by nonlinearities in the electronics. Appears on the screen as graininess texture or diagonal lines over the entire picture.
- *Single-Frequency Modulation Distortion*: Caused by intermodulation products. Its visible effect on the screen consists of horizontal or diagonal bands, broad bands of color variation across the screen.
- *Impulse Noise (Short Duration and High Energy)*: Caused by lightning, car starters, industrial machines, and other sources. Produces spots on the screen and sharp click sounds in the audio.
- *Cochannel Interference*: Caused when two pictures are received at the same time from two different TV transmitters at different locations, but which share the same TV channel. The visible effect occurs on the overlap of the unwanted service and is often accompanied by a "venetian blind" effect and distorted sound.
- *Ghosting*: Caused by multiple transmission paths, resulting from signal reflections. The visible effect consists in the appearance of a delayed version of the picture on the screen.

For digital video, most of the unwanted artifacts result from side effects of the lossy compression techniques used to store or transmit the video using fewer

bits. These are the main types of artifacts that may be present in digital video sequences:

- *Block Edge Effect (or Simply Blockiness)*: Caused by coarse quantization of DCT coefficients in block-based compression schemes (e.g., M-JPEG and MPEG-1), which result in noticeable intensity discontinuities at the boundaries of adjacent blocks in the decoded frame.
- *False Edges (or False Contouring)*: Caused by coarse quantization of amplitude levels. Its visible effect in the frame is the appearance of edges at places where a smooth intensity transition should occur.
- *Ringing*: Caused by the use of ideal filters. Produces a rippling along high-contrast edges.
- *Blurring and Color Bleeding*: Caused by imperfections in compression algorithms. Results in loss of spatial details, running or smearing of color in areas with complex textures or edges.
- *Mosquito Noise*: Caused by imperfections in compression algorithms. Produces fluctuation of luminance/chrominance levels around high-contrast edges or moving video objects.

Video sequences, particularly old movies, can also be subject to other types of artifacts, such as the following:

- *Blotches*: Large, uncorrelated, bright or dark spots, typically present in film material, due to mishandling or aging.
- *Intensity Flicker*: Unnatural temporal fluctuation of frame intensities that do not originate from the original scene.

22.4.2 Interframe Filtering Techniques

Interframe filtering techniques employ a spatiotemporal support region of size $m \times n \times k$, where m and n correspond to the size of a neighborhood in a frame and k denotes the number of frames taken into account. For the special case where $m = n = 1$, the resulting filter is called a *temporal filter*, whereas the case where $k = 1$ corresponds to intraframe filtering. Interframe filtering techniques can be motion compensated, motion adaptive, or neither.

The simplest interframe filtering technique is the spatiotemporal averaging filter that consists of an extension of the basic averaging filter to the spatiotemporal domain. A purely temporal version of the algorithm is known as *frame averaging*. Frame averaging is based on the fact that averaging multiple observations of essentially the same pixel in different frames eliminates noise while resulting in no loss of spatial image resolution. The amount of noise reduction is proportional to the number of frames used, but so is the amount of blurring. In other words, besides the spatial

blurring of 2D averaging filters, we now have a potential motion blur, due to movement of objects between consecutive frames.

There are two solutions to the blurring problem:

- Motion-adaptive, edge-preserving filters, where frame-to-frame motion is treated as "temporal edges."
- Motion-compensated filters, involving: low-pass filters that are applied along the motion trajectory of each pixel, as determined by a motion estimation algorithm.

Motion-compensated filters work under the assumption that the variation of pixel gray levels over any motion trajectory is mainly due to noise. Consequently, noise can be reduced by low-pass filtering over the respective motion trajectory at each pixel. Several motion-compensated filtering techniques have been proposed in the literature. They differ according to the following factors:

- The motion estimation method.
- The support of the filter, where *support* is defined as the union of predetermined spatial neighborhoods centered about the pixel (subpixel) locations along the motion trajectory.
- The filter structure (adaptive versus nonadaptive).

The effectiveness of motion-compensated filters is strongly related to the accuracy of the motion estimates. In the ideal case (perfect motion estimation), direct averaging of image intensities along a motion trajectory provides effective noise reduction. In practice, however, motion estimation is hardly ever perfect due to noise and sudden scene changes, as well as changes in illumination and camera views.

22.5 CASE STUDY: OBJECT SEGMENTATION AND TRACKING IN THE PRESENCE OF COMPLEX BACKGROUND

In this section, we present an example of object segmentation and tracking using MATLAB and the Image Processing Toolbox (IPT).[2] Object detection and tracking are essential steps in many practical applications, for example, video surveillance systems.

The solution presented in Figure 22.12 uses an object segmentation algorithm for video sequences with complex background, originally proposed by Socek et al. [SCM+05] and further refined and redesigned by Ćulibrk et al. [CMS+06,CMS+07]. The output of the segmentation algorithm is a series of binary images (frames), where background pixels are labeled as 0 and any foreground objects are labeled as 1. The

[2]The MATLAB code for this example was developed by Jeremy Jacob and can be downloaded from the book web site.

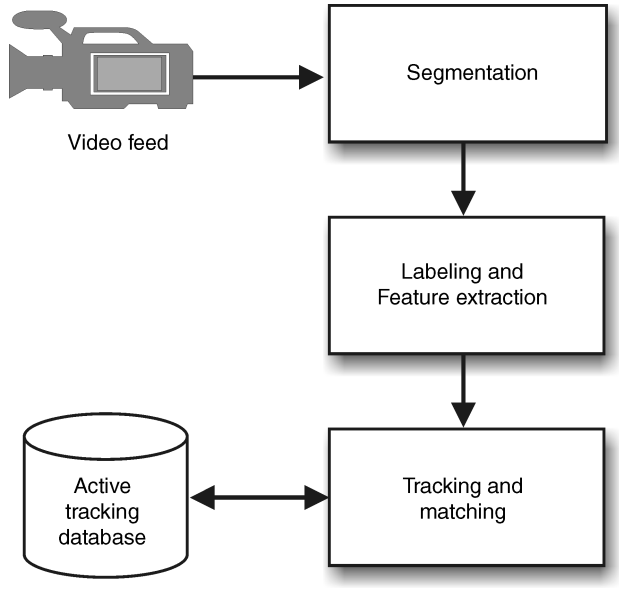

FIGURE 22.12 Object detection and tracking system.

"labeling and feature extraction" stage uses the IPT `bwlabel` function to label each connected component in the foreground of the frame and the IPT `regionprops` function to extract relevant features pertaining to each object, such as the coordinates of the centroid of the object, an index list of the pixels contained in the object, and its bounding box.

At the "tracking and matching" stage, the objects detected in the current frame are compared with the objects currently being tracked from previous frames in order to answer the following questions:

- Is any object new to the scene?
- Has any object left the scene?
- Where did the objects currently being tracked go?

When a new object is discovered (i.e., one for which no current objects match), this object is placed in a "three-frame buffer." The buffer is necessary to dismiss false alarms caused, for example, by noise artifacts introduced in the early stages of segmentation. If the same object is successfully tracked over three frames, it is then removed from the buffer and entered in the "active tracking list." Any object that is currently in this list is considered as an object officially being tracked, and any tracking information collected here will be reported in the final output. Figure 22.13 illustrates the process.

To determine the new position of an existing object A (which is contained in the active tracking list), we consider any object in the current frame whose centroid falls

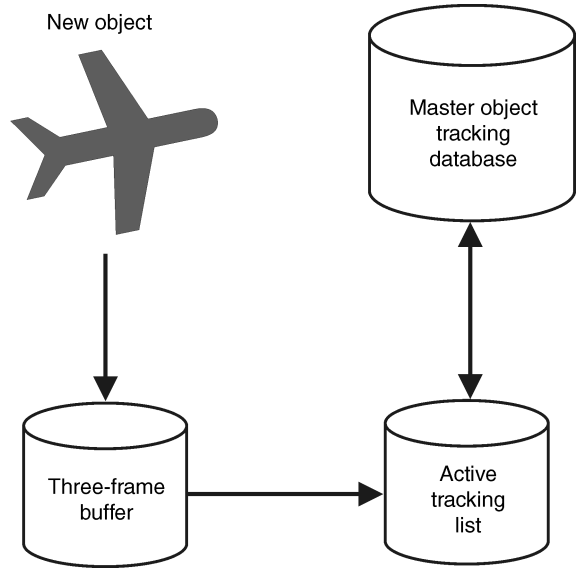

FIGURE 22.13 Keeping track of existing and candidate objects.

within the bounding box of object A as a possible candidate. If only one object meets this criteria, then we consider it a match, run an additional test to ensure that object cannot be considered a possible candidate for any other objects (since we now know what that object is), and update the tracking data for object A. This is illustrated in Figure 22.14. If there are more than one possible candidate for the new position of object A, we then compare object sizes, and that with the closest size to object A is considered a match. Finally, if at any given frame an object cannot be matched up with an object in the subsequent frame, that object is removed from the "active

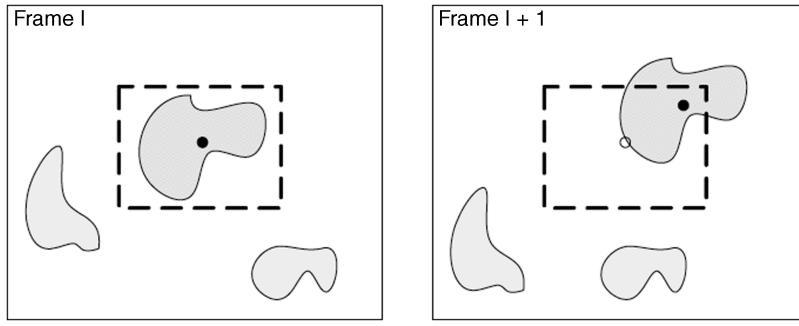

FIGURE 22.14 Updating the coordinates of an existing object.

FIGURE 22.15 Screenshot of the object detection and tracking system, showing the bounding box of an object being tracked, its trajectory since it started being tracked, and its properties (on a separate window).

tracking list," although its history remains in the "master object tracking database" (see Figure 22.12).

For each object, the system keeps track of its size, position, velocity, and color. The size information can be used to separate objects into application- and viewpoint-dependent classes, for example, large, medium, and small. Similarly, brightness and color information can be used to classify objects into categories such as "bright" or "red." Position is reported as the instantaneous pixel position of the centroid of the object within the current frame. Similarly, velocity is reported as an instantaneous change in position between consecutive frames, which can be easily calculated since we know the frame rate of the sequence.

The MATLAB implementation uses a modified version of the `implay` function, which adds the possibility of pausing the sequence and clicking within the bounding box of an object within the scene to open an external window, which displays the tracking information for that object and reports how many objects are being tracked. A screen shot of the movie player—with objects' bounding boxes and trajectory overlapped on the original frame—along with the external window is shown in Figure 22.15.

22.6 TUTORIAL 22.1: BLOCK-BASED MOTION ESTIMATION

Goal

The goal of this tutorial is to explore motion estimation algorithms in MATLAB.

Objectives

- Explore EBMA using both integer-pixel and half-pixel accuracy.
- Explore HBMA using both integer-pixel and half-pixel accuracy.
- Explore the phase correlation method.

What You Will Need

- *MATLAB Functions*: `ebma.m`, `hbma.m`, and `PhaseCorrelation.m`.[3]
- Test files `foreman69.Y` and `foreman72.Y`.

Procedure

Integer-Pixel EBMA

Before we perform the exhaustive block matching algorithm, let us first define a few necessary variables. Here, we will define the anchor frame, target frame, frame dimensions, block size, search field range (p), and finally accuracy (where 1 is for integer-pixel and 2 is for half-pixel).

1. Define initial variables.

```
anchorName = 'foreman69.Y';
targetName = 'foreman72.Y';
frameHeight = 352;
frameWidth = 288;
blockSize = [16,16];
p = 16;
accuracy = 1;
```

Next, we will read in the frame data.

2. Read the frame data.

```
fid = fopen(anchorName,'r');
anchorFrame= fread(fid,[frameHeight,frameWidth]);
anchorFrame = anchorFrame';
fclose(fid);
fid = fopen(targetName,'r');
targetFrame = fread(fid,[frameHeight,frameWidth]);
targetFrame = targetFrame';
fclose(fid);
```

[3]These functions were developed by Jeremy Jacob and can be downloaded from the book web site.

3. Run the function `ebma` and record the time to process the frames in variable `time_full`.

```
tic
[predictedFrame_Full, mv_d, mv_o] = ...
    ebma(targetFrame, anchorFrame, blockSize, p, accuracy);
time_full = toc
```

4. Display the anchor and target frames.

```
figure
subplot(1,2,1), imshow(uint8(anchorFrame)), title('Anchor Frame');
subplot(1,2,2), imshow(uint8(targetFrame)), title('Target Frame');
```

5. Display the motion vectors overlaid on the anchor frame.

```
figure, imshow(uint8(anchorFrame))
hold on
quiver(mv_o(1,:),mv_o(2,:),mv_d(1,:),mv_d(2,:)), ...
    title('Motion vectors: EBMA - Integer-pixel');
hold off
```

6. Display the predicted frame.

```
figure, imshow(uint8(predictedFrame_Full)), ...
    title('Predicted Frame Full-pixel');
```

7. Calculate the error frame by subtracting the predicted frame from the anchor frame.

```
errorFrame = imabsdiff(anchorFrame, predictedFrame_Full);
```

8. Calculate the PSNR by performing the following calculation.

```
PSNR_Full = 10*log10(255*255/mean(mean((errorFrame.^2))))
```

Half-Pixel EBMA

To perform half-pixel EBMA, we must first up-sample the frames.

9. Up-sample the target frame.

```
targetFrame2 = imresize(targetFrame,2,'bilinear');
```

10. Set the accuracy variable appropriately for half-pixel accuracy.

```
accuracy = 2;
```

11. Run the script to perform EBMA with half-pixel accuracy and compute the execution time.

```
tic
[predictedFrame_Half, mv_d, mv_o] = ...
    ebma(targetFrame2, anchorFrame, blockSize, p, accuracy);
time_half = toc
```

12. Compare the time of execution between integer-pixel and half-pixel accuracy.

```
time_half
time_full
```

Question 1 How does the difference in time compare with the additional number of pixels present in the up-sampled frame?

13. Display the motion vectors over the anchor frame.

```
figure, imshow(uint8(anchorFrame)), hold on
quiver(mv_o(1,:),mv_o(2,:),mv_d(1,:),mv_d(2,:)), ...
    title('Motion vectors: EBMA - Half-pixel');
hold off
```

14. Display the predicted frame.

```
figure, imshow(uint8(predictedFrame_Half)), ...
    title('Predicted Frame Half-pixel');
```

15. Calculate the error frame.

```
errorFrame = imabsdiff(anchorFrame, predictedFrame_Half);
```

16. Calculate the PSNR.

```
PSNR_Half = 10*log10(255*255/mean(mean((errorFrame.^2))))
```

Question 2 Compare the PSNR values between integer and half-pixel accuracy. Are the results what you expected?

Question 3 Visually, how do the results compare between integer and half-pixel accuracy (for the predicted frames)?

Multiresolution Motion Estimation: HBMA

To begin implementing HBMA motion estimation in MATLAB, we must first define several variables. Here, we are defining the names of the frames that will be read, the frames' dimensions, block size, start and end ranges, accuracy, and the number of levels to process (in our case, three).

17. Define variables needed to run HBMA.

```
anchorName = 'foreman69.Y';
targetName = 'foreman72.Y';
frameHeight = 352;
frameWidth = 288;
blockSize = [16,16];
rangs = [-32,-32];
range = [32,32];
accuracy = 1;
L = 3;
```

18. Read the frame data.

```
fid = fopen(anchorName,'r');
anchorFrame= fread(fid,[frameHeight,frameWidth]);
fclose(fid);
fid = fopen(targetName,'r');
targetFrame = fread(fid,[frameHeight,frameWidth]);
fclose(fid);
```

We can now run the `hbma` function by calling it with the parameters defined earlier. The function returns the predicted frame, direction of motion vectors, and the respective positions of the motion vectors (i.e., their orientation).

19. Run the `hbma` script and record the time needed to process it.

```
tic
[predict,mv_d,mv_o] = hbma(targetFrame, anchorFrame, blockSize, ...
     rangs, range, accuracy, L);
toc
```

20. Display the anchor and target frames to be able to compare with the predicted frame.

```
figure
subplot(1,2,1), imshow(uint8(anchorFrame')), title('Anchor Frame');
subplot(1,2,2), imshow(uint8(targetFrame')), title('Target Frame');
```

21. Display the motion vectors over the anchor frame.

```
figure, imshow(uint8(anchorFrame'))
hold on
quiver(mv_o(2,:),mv_o(1,:),mv_d(2,:),mv_d(1,:)), ...
    title('Motion vectors');
hold off
```

22. Show the predicted frame.

```
figure, imshow(uint8(predict')), ...
    title('Predicted Frame');
```

23. Calculate the error frame and PSNR.

```
errorFrame = imabsdiff(anchorFrame, predict);
PSNR=10*log10(255*255/mean(mean((errorFrame.^2))))
```

Question 4 How do the motion vectors compare between this and the EBMA technique?

Question 5 How does the quality of the predicted frame obtained with the HBMA algorithm compare with the one generated by the EBMA technique?

Phase Correlation Method

To perform phase correlation, we must first define and read two frames.

24. Read two frames and create the frames array to be used by the script.

```
anchorName = 'foreman69.Y';
targetName = 'foreman72.Y';
frameHeight = 352; frameWidth = 288;
fid = fopen(anchorName,'r');
anchorFrame= fread(fid,[frameHeight,frameWidth]);
anchorFrame = anchorFrame';
fclose(fid);
fid = fopen(targetName,'r');
targetFrame = fread(fid,[frameHeight,frameWidth]);
targetFrame = targetFrame';
fclose(fid);
frame(:,:,1) = anchorFrame;
frame(:,:,2) = targetFrame;
```

25. Run the `PhaseCorrelation.m` script to generate the predicted frame, a phase correlation plot, and a motion vector plot.

Question 6 How does the quality of this technique compare with the previous techniques (EBMA and HBMA)?

22.7 TUTORIAL 22.2: INTRAFRAME AND INTERFRAME FILTERING TECHNIQUES

Goal

The goal of this tutorial is to learn how to perform intraframe and interframe filtering in MATLAB.

Objectives

- Explore the averaging filter, median filter, and Wiener filter applied to video sequences.
- Learn how to use the `noisefilter` and `noisefilter2` functions to apply noise to selected frames in a video sequence and then filter them.

What You Will Need

- *MATLAB Functions*: `readYUV.m`, `noisefilter.m`, and `noisefilter2.m`.[4]
- Test file `football_cif_ori90.yuv`.

Procedure

Intraframe Filtering

To perform intraframe filtering in MATLAB, we will use the `readYUV` function to extract one frame from a YUV video sequence. We will perform filtering in the RGB color space. When using the `readYUV` function, recall that it returns two data structures; one that holds the MATLAB movie structure of the video (`mov`), and another that contains the YUV data for that sequence (`yuv`). The `CDATA` portion of the movie structure is in fact the RGB values for that particular frame—this is where we get the RGB values from.

1. Read in the first frame of the `football_cif_ori90` YUV sequence and extract the RGB data.

[4]These functions were developed by Jeremy Jacob and can be downloaded from the book web site.

```
[mov, yuv] = readYUV('football_cif_ori90.yuv',1,'CIF_PAL');
img = mov(1).cdata;
```

Before filtering, let us add artificial noise; this will give us something to filter out.

2. Create two noisy images: one with salt and pepper noise and another with Gaussian noise.

```
noisySP = imnoise(img, 'salt & pepper');
noisyGauss = imnoise(img,'gaussian');
```

3. Display the frames for later comparisons.

```
figure
subplot(1,3,1), imshow(img), title('Original frame');
subplot(1,3,2), imshow(noisySP), title('Salt & Pepper noise');
subplot(1,3,3), imshow(noisyGauss), title('Gaussian noise');
```

Our first filter is an averaging filter. Here, the pixel in question is simply taken as the average of the pixels in the given window size surrounding that pixel.

4. Apply an averaging filter to each noisy image.

```
filt = fspecial('average');
avgFilterFrame1 = imfilter(noisyGauss,filt);
avgFilterFrame2 = imfilter(noisySP,filt);
```

5. Display the results.

```
figure
subplot(1,3,1), imshow(img), title('Original frame');
subplot(1,3,2), imshow(avgFilterFrame2), ...
    title('Salt & Pepper averaged');
subplot(1,3,3), imshow(avgFilterFrame1), ...
    title('Gaussian averaged');
```

We will now filter a sequence of movie frames using the `noisefilter` function.

6. Clear all variables and close all figures.
7. Load 30 frames of the `football_cif_ori90.yuv` sequence.

```
[mov, yuv] = readYUV('football_cif_ori90.yuv',30,'CIF_PAL');
```

8. Add noise to all frames in the sequence and filter.

```
mov2 = noisefilter(mov,'salt & pepper','average',[]);
```

9. Play the resulting movie.

```
implay(mov2)
```

10. Add noise to only a select number of frames.

```
mov3 = noisefilter(mov,'salt & pepper','average',[5 10 20]);
```

11. Play the resulting movie.

```
implay(mov3)
```

Question 1 Was the noise noticeable even though the frames were filtered (for both `mov2` and `mov3`)?

Question 2 Filter the same sequences using the median filter. Is the noise noticeable?

Interframe Filtering Techniques

To perform interframe filtering in MATLAB, we will use the function `noisefilter2`. This is similar to the function `noisefilter` used earlier in this tutorial in that it will apply noise to designated frames (defined in the function call). It differs in the way the frames are filtered; frame data will be averaged over the temporal domain.

Let us perform interframe filtering on a video sequence to see its effect.

12. Load the video sequence.

```
[mov, yuv] = readYUV('football_cif_ori90.yuv',50,'CIF_PAL');
```

13. Apply noise to all frames, and average over three frames.

```
mov2 = noisefilter2(mov,'salt & pepper',[],3);
```

14. Play the resulting movie.

```
implay(mov2)
```

Question 3 What is the effect of the noise once the interframe filtering is applied (after the third frame)?

Question 4 Filter the video sequence again using a higher number of frames to be used in the filtering process. How does this affect noise? How does it affect image quality?

WHAT HAVE WE LEARNED?

- Motion estimation is the process of determining how much relative motion (between camera and scene) has occurred between two video frames. This information is usually encoded using motion vectors. Motion estimation is a required step for many video processing algorithms, from compression to spatiotemporal filtering.
- There are numerous motion estimation techniques and algorithms in the literature. They differ in the way they represent motion, the level of granularity used (from pixels to blocks to entire frames), the criteria used to determine whether motion was present, and the mathematical function that should be optimized as a result.
- Motion compensation is the process of using the results of the motion estimation stage (usually encoded in the form of motion vectors) to perform another video processing operation (e.g., spatiotemporal filtering) in such a way that takes into account the motion of relevant portions of the frame over time.
- Video filtering techniques usually fall into two categories: intraframe and interframe. The former treats one frame at a time and has strong resemblance to comparable image filtering techniques. The latter processes several (e.g., three) consecutive frames at a time and can benefit from results generated by a motion estimation step.

LEARN MORE ABOUT IT

- The topic of motion estimation has received book-length coverage (e.g., in [FGW96]), and appeared in many surveys (e.g., [SK99,HCT+06]).
- Chapters 6 and 7 of [WOZ02] discuss 2D and 3D motion estimation in great detail.
- Chapter 3.10 of [Bov00a] is entirely devoted to motion detection and estimation.
- Motion compensation techniques, especially the phase correlation method, are described in [Wat94a].
- Chapter 10 of [Woo06] covers intraframe and interframe filtering techniques, including many examples of motion-compensated video filtering.
- The topic of motion-compensated filtering—theory and algorithms—is covered in great depth in Chapters 13 and 14 of [Tek95].

- Section 16.2 of [SHB08] provides a good introduction to optical flow and its application in motion analysis.
- Chapter 5 of [Tek95] presents motion estimation using optical flow in great detail.
- Sections 16.5 and 16.6 of [SHB08] discuss contemporary methods used in video tracking solutions.
- Chapter 3.11 of [Bov00a] presents a detailed discussion on video enhancement and restoration techniques, including blotch detection and removal and intensity flicker correction.

22.8 PROBLEMS

22.1 Assume a very simple motion detection algorithm, in which to estimate the motion in a scene you examine the intensity of two captured video frames, $f(n_1, n_2, t_1)$ and $f(n_1, n_2, t_2)$ captured at t_1 and t_2 respectively, and you compute the absolute difference between them: $d(n_1, n_2) = |f(n_1, n_2, t_2) - f(n_1, n_2, t_1)|$.

(a) If the frame difference is nonzero, can you say that motion has occurred in the scene? Explain.

(b) If the frame difference is exactly zero, can you say that no motion has occurred in the scene? Explain.

22.2 Consider the phase correlation method for motion estimation and answer the following questions:

(a) What is this method's main strength?

(b) What is its principal weakness and how can it be overcome?

22.3 Answer the following questions (in your own words) about motion estimation.

(a) Why is motion estimation in video sequences such an *important* problem? (Select two–three specific video processing operations that benefit from or rely on motion estimation and describe them.)

(b) Why is motion estimation in video sequences such a *difficult* problem? (Select two–three specific problems and describe them.)

(c) What is the chief disadvantage of the exhaustive block matching algorithm?

22.4 Write a MATLAB script or function that will perform the three-step motion estimation algorithm and test it using frames 69 and 72 of the foreman sequence.

22.5 Write a MATLAB script or function that will perform the 2D log motion estimation algorithm and test it using frames 69 and 72 of the foreman sequence.