Concise CV

Chapter-4

3-D Motion and 2-D Optical Flow

- In a sequence of image with time difference δt b/w to subsequent image. For e.g we have $\delta(t) = 1/30$ sec if the image is recorded at a frequency of 30 Hz also called 30 frames/sec

→ Let $I(x, y, t)$ denotes recorded frame at time t for pixed locations at x, y

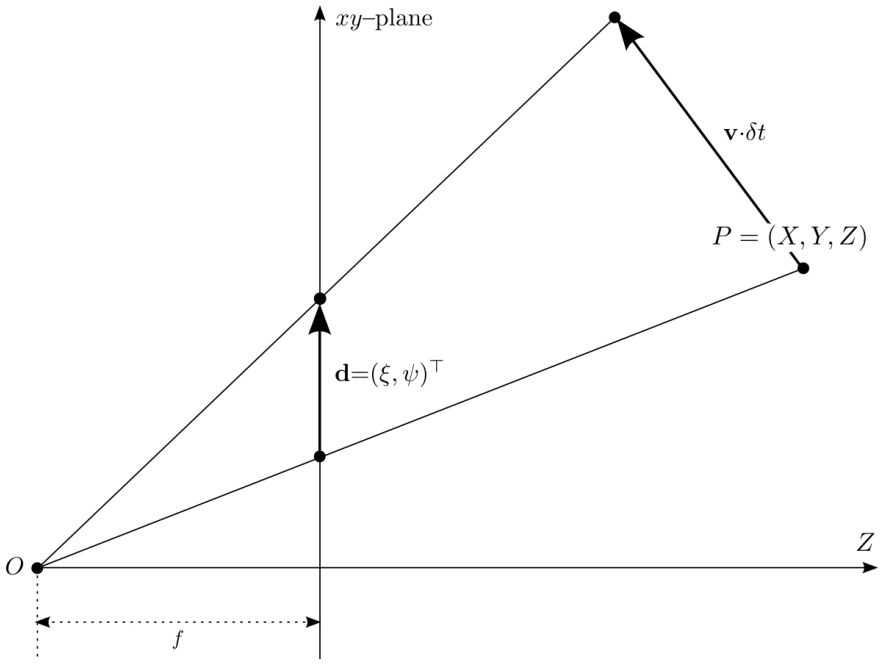

Fig. 4.1 Projection of velocity **v** into a displacement **d** in the image plane

2D motion

→ 3D movement of P b/w t & $\delta(t)$ and $(t+1)\,\delta(t)$ with velocity $V = (V_x, V_y, V_z)^T$

→ 3D motion from point $P = (X, Y, Z)$ to point $(X + V_x\,\delta(t),\ Y + V_y\,\delta(t),\ Z + V_z\,\delta(t))$

→ The vector $d = (\xi, \psi)^T$ is the projection of 3D motion of point P b/w image $I(:; t)$ to $I(:; t+1)$. The 2D motion is defined as the local displacement of pixel originally at P assuming we know its 3D motion is optical flow

– The visible displacement in 2D motion $u = (u, v)^T$. It starts at pixel location $p = (x, y)$ and ends at $p = (x+u, y+v)$

→ Optical flow aims at the estimation of 2D Motion.

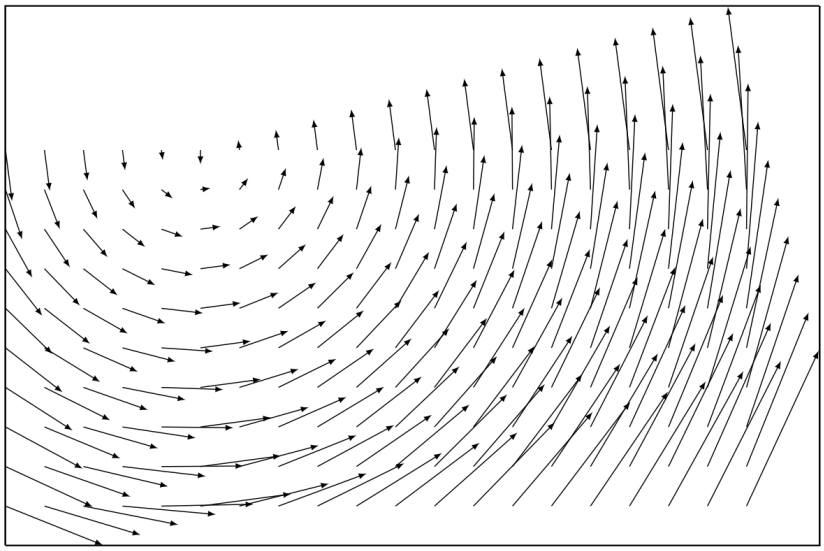

Fig. 4.3 A vector field showing the rotation of a rectangle. Motion vectors start at the position at time t and end at the position of the rotated start point at time $t + 1$

Vector Fields We like to understand the motion of 3D objects by analysing projections of many their surface points $P = (X, Y, Z)$. The results should be consistent with object movements or with the shape of moving rigid objects.

2D motion vectors form a *vector field*. See Fig. 4.3 for an example, illustrating a rotating rectangle (around a fixpoint) in a plane parallel to the image plane.

A rigid body simplifies the analysis of 2D motion vector fields, but even then those vector fields are not easy to read. As a graphical exercise, you may generate visualizations of 2D motion vector fields for simple 3D shapes such as a cube or other simple polyhedra. It is difficult to infer the shape of the polyhedron when just looking at the motion field.

→ A computed motion vector field is called dense if it contains motion vector at each pixel location and otherwise sparse

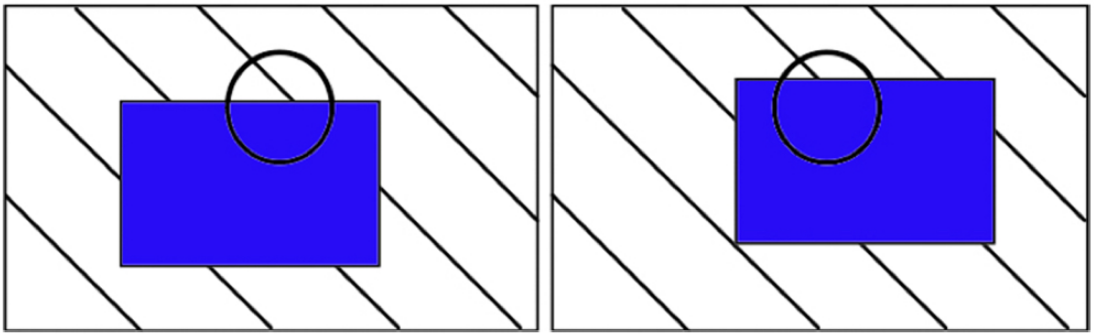

Fig. 4.6 Seeing only both circular windows at time t (*left*) and time $t+1$ (*right*), we conclude an upward shift and miss the shift diagonally towards the *upper right corner*

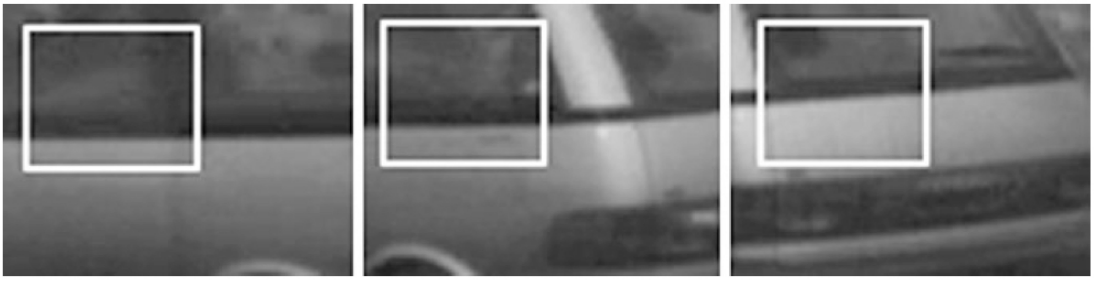

Fig. 4.7 Three images taken at times t, $t+1$, and $t+2$. For the *inner rectangles*, we may conclude an upward translation with a minor rotation, but the three images clearly indicate a motion of this car to the *left*

By analysing projected images, we observe a limited area of visible motion, defined by the aperture of the camera, or by the algorithm used for motion analysis. Such an algorithm checks, in general, only a limited neighbourhood of a pixel for deriving a conclusion.

The Aperture Problem Recall a situation in a train waiting for departure, looking out of the window, and believing that your train started to move, but it was actually the train on the next track that was moving. An aperture problem is caused by a limited view on a dynamic scene. The aperture problem adds further uncertainties to motion estimation.

Figure 4.6 illustrates a situation where an algorithm processes an image and "sees" around the current pixel only the sketched circular area and nothing else. We may conclude that there is an upward move; the diagonal motion component is not apparent in the circular window.

For a real-world example, see Fig. 4.7. A car drives around a roundabout, to the left and away from the camera. We only see a limited aperture. If we even only see the inner rectangles, then we may conclude an upward shift.

However, possibly there is actually something else happening, for example, the car may not move by itself, but is carried on a trailer, or, maybe, the images show some kind of mobile screen driven around on a truck. Increasing the aperture will help, but it remains that we possibly have incomplete information.

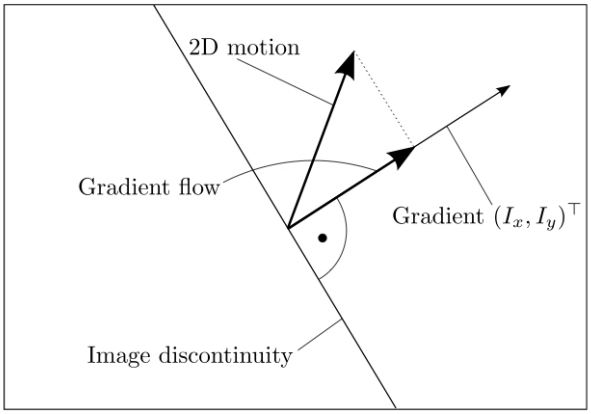

Fig. 4.8 Illustration of a gradient flow. The true 2D motion **d** goes diagonally up, but the identified motion is the projection of **d** onto the gradient vector

Gradient Flow Due to the aperture problem, a local method typically detects a *gradient flow*, which is the projection of the true 2D motion onto the gradient at the given pixel. The 2D gradient with respect to coordinates x and y,

$$\nabla_{x,y} I = \left[I_x(x, y, t), I_y(x, y, t) \right]^\top \tag{4.1}$$

is orthogonal to the straight image discontinuity (an edge) as illustrated in Fig. 4.8, assuming that the image intensities decrease from the region on the left to the region on the right. Recall that I_x and I_y in (4.1) denote the partial derivatives of the frame $I(\cdot, \cdot, t)$ with respect to x and y, respectively.

The result may improve (i.e. going more towards the true 2D motion) by taking the values at adjacent pixels into account. For example, an object corner may support the calculation of the correct 2D motion.

The Horn Schunck Algorithm

Relationship between $I(:,:,t)$ and $I(:,:,t+1)$

using first order 3-D Taylor expansion we obtain following:

$I[x+\delta x, y+\delta y, t+\delta t)$

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

e represents all second and higher order derivatives

We assume $e=0$ means our function behaves as linear for small values of $\delta(x)$, $\delta(y)$ and $\delta(t)$

Using intensity consistency assumption (ICA)

$$I(x+\delta(x), y+\delta(y), t+\delta(t)) = I(x,y,t)$$

before and after the motion

$$\delta(x)\frac{dI}{\partial x}(x,y,t) + \delta(y)\frac{dI}{dy}(x,y,t)$$

$$+\delta(t)\frac{dI}{dt}(x,y,t) = 0$$

Dividing the above equation by $\delta(t)$ we get

$$\frac{\delta(x)}{\delta(t)}\frac{\partial I}{\partial x}(x,y,t) + \frac{\delta(y)}{\delta(t)}\frac{\partial I}{dy}(x,y,t)$$

$$+\frac{\delta(t)}{\delta(t)}\frac{\partial I}{\partial t}(x,y,t) = 0$$

The changes in x and y coordinates represent the optical flow $u(x,y,t)$

$$= (u(x,y,t), v(x,y,t))^T$$

$$0 = u(x,y,t)\frac{\partial I}{\partial x}(x,y,t) + v(x,y,t)\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t}(x,y,t)$$

↑

Optical flow eqn on Horn-schunk constraint

Assumption of the above derivations is constant intensity and small steps

The uv Velocity Space We express (4.6) in short form as follows by ignoring the coordinates (x, y, t):

$$-I_t = u \cdot I_x + v \cdot I_y = \mathbf{u} \cdot \nabla_{x,y} I \tag{4.7}$$

Here, notation I_x and I_y is short for the partial derivatives (as used above) with respect to x- or y-coordinates, respectively, and I_t for the one with respect to t. The scalar $\mathbf{u} \cdot \nabla_{x,y} I$ is the *dot product* of the optical flow vector times the gradient vector (i.e. partial derivatives only with respect to x and y, not for t).

Insert 4.4 (Inner or Dot Vector Product) *Consider two vectors* $\mathbf{a} = (a_1, a_2, \ldots, a_n)^\top$ *and* $\mathbf{b} = (b_1, b_2, \ldots, b_n)^\top$ *in the Euclidean space* $\mathbb{R}^n$. *The dot product, also called the inner product, of both vectors is defined as*

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

It satisfies the property

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \cdot \|\mathbf{b}\|_2 \cdot \cos\alpha$$

where $\|\mathbf{a}\|_2$ *and* $\|\mathbf{b}\|_2$ *are the magnitudes of both vectors, for example*

$$\|\mathbf{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

and α *is the angle between both vectors, with* $0 \leq \alpha \leq \pi$.

I_x, I_y and I_t play the role of parameters in (4.7). They are estimated in the given frames by discrete approximations of derivatives (e.g., Sobel values S_x and S_y for I_x and I_y). Optic flow components u and v are the unknowns in (4.7). Thus, this equation defines a straight line in the *uv velocity space*. See Fig. 4.9.

By (4.7) we know that the optical flow (u, v) for the considered pixel (x, y, t) is a point on this straight line, but we do not know yet which one.

Labels are assigned to all the pixel in Ω by a labelling function.

$f: \Omega \longrightarrow L$

Solving a labelling function to identify the labelling f that approximates somehow the given optimum.

First constraint

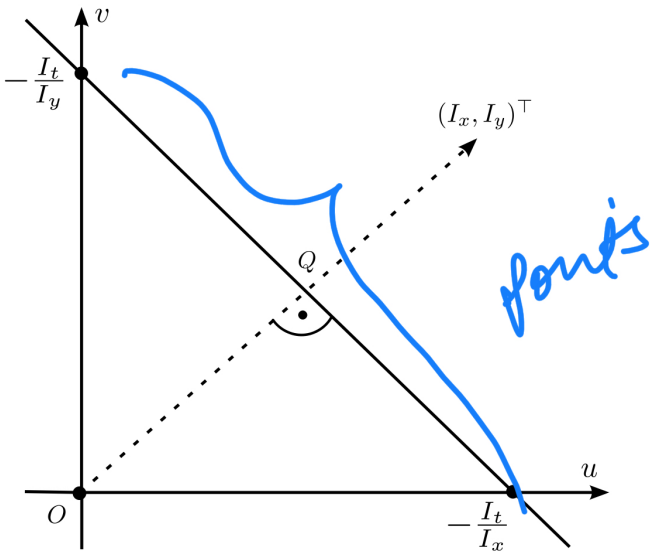

Fig. 4.9 The *straight line* $-I_t = u \cdot I_x + v \cdot I_y$ in the uv velocity space. The point Q is discussed in Example 4.1

points

Let f be a labelling function that assigns a label (u,v) to each pixel $p \in \Omega$ in an image $I(\cdot,\cdot,t)$

Edata (f) = $\sum_{\Omega}$ [$I_t + u \cdot I_x + v \cdot I_y$]

f

Energy dat with

$u I_x + v I_y + I_t = 0$ in ideal case

for pixel $p \in \Omega$ in image

$I(\cdot, \cdot, t)$

Second - Constraint: Spatial Motion

Consistency

Second Constraint—Spatial Motion Constancy Additionally to (4.7), we formulate a second constraint for solutions (u, v), hoping that this leads to a unique solution on the straight line $-I_t = u \cdot I_x + v \cdot I_y$. There are many different options for formulating such a second constraint.

We assume motion constancy within pixel neighbourhoods at time t, which means that adjacent pixels in $I(\cdot, \cdot, t)$ have about the same optical flow vectors. For a function that is nearly constant in local neighbourhoods, its first derivatives will be close to zero. Thus, motion constancy can be formulated as a *smoothness constraint* that the sum of the squares of first-order derivatives needs to be minimized for all pixels $p \in \Omega$ in $I(\cdot, \cdot, t)$. Again, we keep it short without writing arguments (x, y, t).

Let f be the labelling function again that assigns a label (u, v) to each pixel $p \in \Omega$ in $I(\cdot, \cdot, t)$. The *smoothness error*, also called the *smoothness energy*, is defined as

$$E_{smooth}(f) = \sum_{\Omega} \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \qquad (4.10)$$

optimization problem

$E_{total}(f) = E_{data}(f) + \lambda E_{smooth}(f)$

$0 < \lambda < 1$ ← to avoid strong smoothening

We search for an optimum f in the set of all possible labellings, which defines a *total variation* (TV). The used smoothing error term applies squared penalties in the L_2 sense. Altogether, this defines a TVL_2 optimization problem. A value of $\lambda > 1$ would give smoothness a higher weight than the Horn–Schunck constraint (i.e., the data term), and this is not recommended. A value such as $\lambda = 0.1$ is often a good initial guess, allowing smoothness a relatively small impact only.

Insert 4.5 (Least-Square Error Optimization) *Least-square error* (*LSE*) *optimization follows the following standard scheme*:

1. *Define an error or energy function.*
2. *Calculate the derivatives of this function with respect to all unknown parameters.*
3. *Set the derivatives equal to zero and solve this equation system with respect to the unknowns. The result defines a stationary point, which can only be a local minimum, a saddle point, or a local maximum.*

For simplicity, we assume all pixel have their adjacent pixel in the image itself

Derivatives in smoothness error term by simple asymmetric difference is

$$E_{smooth}(f) = \sum_{x,y} (u_{x+1,y} - u_{xy})^2$$

$$(u_{x,y+1} - u_{xy})^2 + (v_{x,y+1} - v_{xy})^2$$

$$+ (v_{x+1,y} - v_{xy})^2$$

$$\frac{\partial E_{data}(u,v)}{\partial u_{x,y}} = 2[I_x(x,y)u_{xy}$$

$$+ I_y(x,y)v_{xy} + I_t(x,y)]$$

$$\cdot I_x(x,y)$$

$$\frac{\partial E_{data}(u,v)}{\partial v_{x,y}} = 2[I_x(x,y)u_{xy}$$

$$+ I_y(x,y)v_{xy} + I_t(x,y)]$$

$$I_y(x,y)$$

The partial derivative of $E_{smooth}(u,v)$ are equal to

$$\frac{\partial E_{smooth}}{\partial u_{xy}}(u,v) = -2\left[(u_{x+1,y} - u_{x,y}) + u_{x,y+1} - u_{xy}\right]$$

$$+2\left[(u_{x,y} - u_{x-1,y}) + (u_{xy} - u_{x,y-1})\right]$$

$$= 2\left[(u_{xy} - u_{x+1,y}) + (u_{xy} - u_{x,y+1})\right.$$

$$\left. + (u_{xy} - u_{x-1,y}) + (u_{xy} - u_{x,y-1})\right]$$

These are the only terms containing the unknown u_{xy}. We simplify this expression and obtain

$$\frac{1}{4} \cdot \frac{\partial E_{smooth}}{\partial u_{xy}}(u, v) = 2\left[u_{xy} - \left[\frac{1}{4}(u_{i+1,j} + u_{x,y+1} + u_{i-1,j} + u_{x,y-1})\right]\right] \quad (4.16)$$

Using $\bar{u}_{xy}$ for the mean value of 4-adjacent pixels, we have that

$$\frac{1}{4}\frac{\partial E_{smooth}}{\partial u_{xy}}(u, v) = 2[u_{xy} - \bar{u}_{xy}] \quad (4.17)$$

$$\frac{1}{4}\frac{\partial E_{smooth}}{\partial v_{xy}}(u, v) = 2[v_{xy} - \bar{v}_{xy}] \quad (4.18)$$

Equation (4.18) follows analogously to the provided calculations.

Altogether, using λ instead of $\lambda/4$, after setting derivatives equal to zero, we arrive at the equation system

$$\begin{aligned} 0 &= \lambda[u_{xy} - \bar{u}_{xy}] \\ &\quad + \left[I_x(x, y)u_{xy} + I_y(x, y)v_{xy} + I_t(x, y)\right]I_x(x, y) \quad (4.19) \\ 0 &= \lambda[v_{xy} - \bar{v}_{xy}] \\ &\quad + \left[I_x(x, y)u_{xy} + I_y(x, y)v_{xy} + I_t(x, y)\right]I_y(x, y) \quad (4.20) \end{aligned}$$

This is a discrete scheme for minimizing equation (4.11). This is a linear equational system for $2N_{cols}N_{rows}$ unknowns u_{xy} and v_{xy}. The values of I_x, I_y, and I_t are estimated based on given image data.

Iterative Solution Scheme This equation system also contains the means $\bar{u}_{xy}$ and $\bar{v}_{xy}$, which are calculated based on the values of those unknowns. The dependency between the unknowns u_{xy} and v_{xy} and means $\bar{u}_{xy}$ and $\bar{v}_{xy}$ within those equations

is actually of benefit. This allows us to define an *iterative scheme* (an example of a *Jacobi method*; for C.G.J. Jacobi, see Insert 5.9), starting with some initial values:

1. *Initialization step:* We initialize the values u_{xy}^0 and v_{xy}^0.
2. *Iteration Step 0:* Calculate the means $\bar{u}_{xy}^0$ and $\bar{v}_{xy}^0$ using the initial values and calculate the values u_{xy}^1 and v_{xy}^1.
3. *Iteration Step n:* Use the values u_{xy}^n and v_{xy}^n to compute the means $\bar{u}_{xy}^n$ and $\bar{v}_{xy}^n$; use those data to calculate the values u_{xy}^{n+1} and v_{xy}^{n+1}.

Proceed for $n \geq 1$ until a stop criterion is satisfied.

We skip the details of solving the equation system defined by (4.19) and (4.20). This is a standard linear algebra. The solution is as follows:

$$u_{xy}^{n+1} = \bar{u}_{xy}^n - I_x(x,y) \cdot \frac{I_x(x,y)\bar{u}_{xy}^n + I_y(x,y)\bar{v}_{xy}^n + I_t(x,y)}{\lambda^2 + I_x^2(x,y) + I_y^2(x,y)} \tag{4.21}$$

$$v_{xy}^{n+1} = \bar{v}_{xy}^n - I_y(x,y) \cdot \frac{I_x(x,y)\bar{u}_{xy}^n + I_y(x,y)\bar{v}_{xy}^n + I_t(x,y)}{\lambda^2 + I_x^2(x,y) + I_y^2(x,y)} \tag{4.22}$$

Now we are ready to discuss the algorithm.

The Algorithm

Let

$$\alpha(x,y,n) = \frac{I_x(x,y)\bar{u}_{xy}^n + I_y(x,y)\bar{v}_{xy}^n + I_t(x,y)}{\lambda^2 + I_x^2(x,y) + I_y^2(x,y)} \tag{4.23}$$

$\bar{u}$, $\bar{v}$ denotes the mean of the 4-adjacent pixel

u_{xy} and v_{xy} is initialized to 0 as suggested in Horn Schunk Algorithm algorithm. This allows us to have non-zero values u_{xy} and v_{xy} as long as $I_x(x,y) \cdot I_t(x,y)$ and $I_y(x,y) \cdot I_t(x,y)$ are not equal to 0 at all pixel locations

The original Horn Schunck algorithm used the following asymetric approximation for I_x, I_y and I_t

```
1: for y = 1 to N_rows do
2:     for x = 1 to N_cols do
3:         Compute I_x(x, y), I_y(x, y), and I_t(x, y) ;
4:         Initialize u(x, y) and v(x, y) (in even arrays);
5:     end for
6: end for
7: Select weight factor λ; select T > 1; set n = 1;
8: while n ≤ T do
9:     for y = 1 to N_rows do
10:        for x = 1 to N_cols {in alternation for even or odd arrays} do
11:            Compute α(x, y, n);
12:            Compute u(x, y) = ū − α(x, y, n) · I_x(x, y, t) ;
13:            Compute v(x, y) = v̄ − α(x, y, n) · I_y(x, y, t) ;
14:        end for
15:    end for
16:    n := n + 1;
17: end while
```

Fig. 4.10 Horn–Schunck algorithm

$$
\begin{aligned}
I_x(x,y,t) = \frac{1}{4}\big[&I(x+1,y,t) + I(x+1,y,t+1) \\
&+ I(x+1,y+1,t) + I(x+1,y+1,t+1)\big] \\
&- \frac{1}{4}\big[I(x,y,t) + I(x,y,t+1) + I(x,y+1,t) + I(x,y+1,t+1)\big]
\end{aligned}
\tag{4.24}
$$

$$
\begin{aligned}
I_y(x,y,t) = \frac{1}{4}\big[&I(x,y+1,t) + I(x,y+1,t+1) \\
&+ I(x+1,y+1,t) + I(x+1,y+1,t+1)\big] \\
&- \frac{1}{4}\big[I(x,y,t) + I(x,y,t+1) + I(x+1,y,t) + I(x+1,y,t+1)\big]
\end{aligned}
\tag{4.25}
$$

$$
\begin{aligned}
I_t(x,y,t) = \frac{1}{4}\big[&I(x,y,t+1) + I(x,y+1,t+1) \\
&+ I(x+1,y,t+1) + I(x+1,y+1,t+1)\big] \\
&- \frac{1}{4}\big[I(x,y,t) + I(x,y+1,t) + I(x+1,y,t) + I(x+1,y+1,t)\big]
\end{aligned}
\tag{4.26}
$$

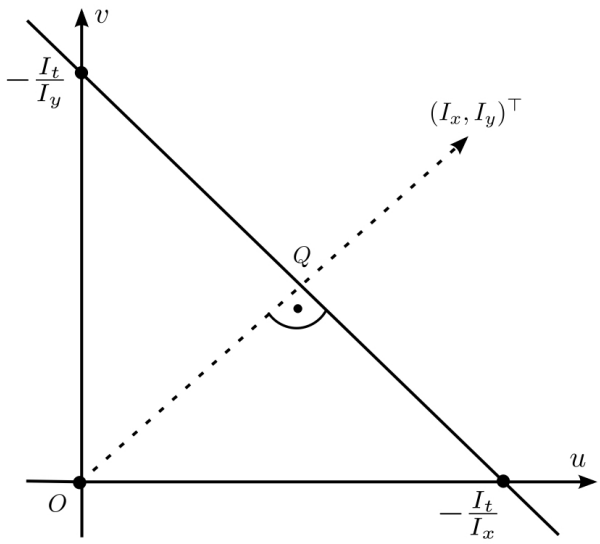

Fig. 4.9 The *straight line* $-I_t = u \cdot I_x + v \cdot I_y$ in the uv velocity space. The point Q is discussed in Example 4.1

First Constraint Let f be a labelling function that assigns a label (u, v) to each pixel $p \in \Omega$ in an image $I(\cdot, \cdot, t)$. Due to (4.7), we are interested in a solution f that minimizes the *data error*, also called the *data energy*,

The unknown optical flow $u = [u, v]^T$ at pixel (x, y) in the image $I(\cdot, \cdot, t)$, relative to the image $I(\cdot, \cdot, t+1)$ can be any vector starting at origin O in fig 4.9 and ending somewhere at the straight line. For initialization you may take a point on straight line, which is defined by the optical flow eqn in the uv velocity space;

instead of simply taking 0 for initialization. For eg we can choose a point close to origin as S as shown in the image

The straight line $I_t = u \cdot I_x + v \cdot I_y$ intersect the u and v axis at the points $(-I_t/I_x, 0)$ $(0, -I_t/I_y)$ respectively. Thus, the vector

$$[-I_t/I_x, 0]^T - [0, -I_t/I_y]^T$$

$$= [-I_t/I_x, I_t/I_y]^T$$ is parallel to this straight line.

Let a be the vector from 0 to S then dot product as $\cos\frac{\pi}{2} = 0$ is

$$\mathbf{a} \cdot [-I_t/I_x, I_t/I_y]^\top = 0 \tag{4.28}$$

From this equation and $\mathbf{a} = [a_x, a_y]^\top$ it follows that

$$a_x \cdot (-I_t/I_x) + a_y \cdot (I_t/I_y) = 0 \tag{4.29}$$

and

$$I_t(a_x \cdot I_y) = I_t(a_y \cdot I_x) \tag{4.30}$$

Assuming a change (i.e., $I_t \neq 0$) at the considered pixel location $p = (x, y)$, it follows that $a_x I_y = a_y I_x$ and

$$\mathbf{a} = c \cdot \mathbf{g}^\circ \tag{4.31}$$

for some constant $c \neq 0$, where $\mathbf{g}^\circ$ is the unit vector of the gradient $\mathbf{g} = [I_x, I_y]^\top$. Thus, $\mathbf{a}$ is a multiple of the gradient $\mathbf{g}$, as indicated in Fig. 4.9.[1]

The vector $\mathbf{a}$ satisfies the optical flow equation $\mathbf{u} \cdot \mathbf{g} = -I_t$. It follows that

$$c \cdot \mathbf{g}^\circ \cdot \mathbf{g} = c \cdot \|\mathbf{g}\|_2 = -I_t \tag{4.32}$$

Thus, we have c and altogether

$$\mathbf{a} = -\frac{I_t}{\|\mathbf{g}\|_2}\mathbf{g}^\circ \tag{4.33}$$

We can use the point Q, as defined by vector $\mathbf{a}$, for initializing the u and v arrays prior to the iteration. That means we start with the *gradient flow*, as represented by $\mathbf{a}$. The algorithm may move away from the gradient flow in subsequent iterations due to the influence of values in the neighbourhood of the considered pixel location.

For approximating I_x, I_y, and I_t, you may also try a very simple (e.g. two-pixel) approximation. The algorithm is robust—but results are erroneous, often mainly due to the intensity constancy assumption (ICA), but there is also a questionable impact of the smoothness constraint (e.g. at motion discontinuities in the considered frames) and, of course, the aperture problem. The number T of iterations can be kept fairly small (say, $T = 7$) because there are typically no real improvements later on.

A *pyramidal Horn–Schunck algorithm* uses image pyramids as discussed in Sect. 2.2.2. Processing starts at a selected level (of lower resolution) images. The obtained results are used for initializing optical flow values at a lower level (of higher resolution). This can be repeated until the full resolution level of the original frames is reached.

Lucas Kanade Algorithm

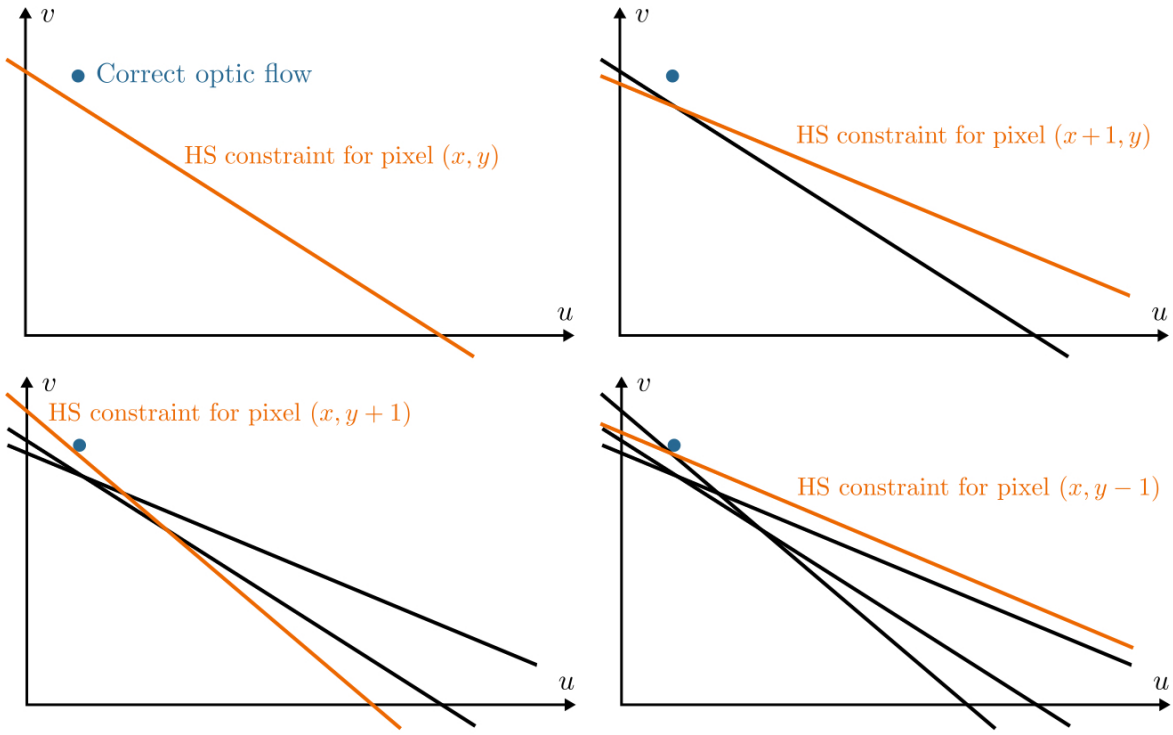

Fig. 4.13 Taking more and more pixels in a local neighbourhood into account by analysing intersection points of lines defined by the optical flow equations for those pixels

The optical flow eqn specifies a straight line $u \cdot I_x + v \cdot I_y + I_t = 0$ in uv velocity space for each pixel $p \in \Omega$. Consider all straight lines defined by pixel in local neighbourhood Assuming they are not parallel but defined by the same 2D motion, they

intersect somehow close to the true 2D motion.

Linear least-square solution

The Lucas–Kanade optical flow algorithm applies a linear least-squares method for analysing the intersections of more than two straight lines.

Insert 4.8 (Method of Least Squares) *This method applies in cases of overdetermined equational systems. The task is to find a solution that minimizes the sum of squared errors (called residuals) caused by this solution for each of the equations.*

For example, we only have n unknowns but $m > n$ linear equations for those; in this case we use a linear least-squares method.

We start with having only two straight lines (i.e. pixel location p_1 and one adjacent location p_2). Assume that 2D motion at both adjacent pixels p_1 and p_2 is identical and equals $\mathbf{u}$. Also assume that we have two different unit gradients $\mathbf{g}_1^\circ$ and $\mathbf{g}_2^\circ$ at those two pixels.

Consider the optical flow equation $\mathbf{u}^\top \cdot \mathbf{g}^\circ = -\frac{I_t}{\|\mathbf{g}\|_2}$ at both pixels (in the formulation as introduced in Example 4.1). We have two equations in two unknowns u and v for $\mathbf{u} = (u, v)^\top$:

$$\mathbf{u}^\top \cdot \mathbf{g}_1^\circ(p_1) = -\frac{I_t}{\|\mathbf{g}\|_2}(p_1) \tag{4.34}$$

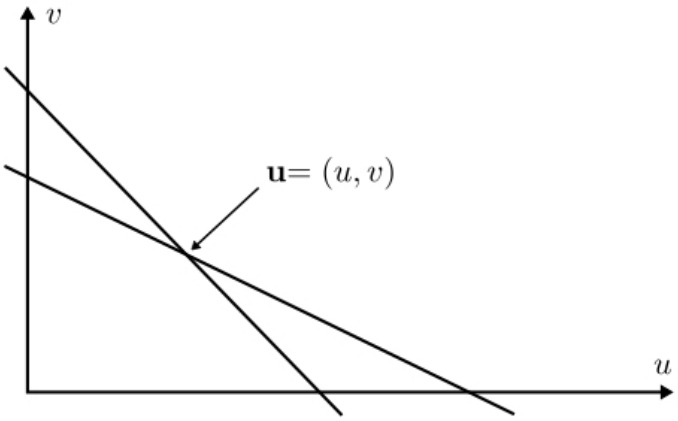

Fig. 4.14 Simple case when considering only two pixels

$$\mathbf{u}^\top \cdot \mathbf{g}_2^\circ(p_2) = -\frac{I_t}{\|\mathbf{g}\|_2}(p_2) \tag{4.35}$$

Using b_i on the right-hand side, this can also be written in the form of a linear equation system with two unknowns u and v:

$$u g_{x1} + v g_{y1} = b_1 \tag{4.36}$$

$$u g_{x2} + v g_{y2} = b_2 \tag{4.37}$$

for unit vectors $\mathbf{g}_1^\circ = [g_{x1}, g_{y1}]^\top$ and $\mathbf{g}_2^\circ = [g_{x2}, g_{y2}]^\top$. We write these equations in matrix form:

$$\begin{bmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{4.38}$$

We can solve this system if the matrix on the left is invertible (i.e. non-singular):

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \end{bmatrix}^{-1} \cdot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{4.39}$$

This is the case where we do not have an overdetermined equational system. We have the intersection of two lines defined by the optical flow equations for p_1 and p_2; see Fig. 4.14.

There are errors involved when estimating I_x, I_y, and I_t, and image data are noisy anyway, so it is best to solve for $\mathbf{u}$ in the least-squares sense by considering a neighbourhood of $k > 2$ pixels, thus having an overdetermined equation system. The neighbourhood should be not too large because we assume that all pixels in this neighbourhood have the same 2D motion $\mathbf{u}$. This leads to the overdetermined linear equation system

$$\begin{bmatrix} g_{x1} & g_{y1} \\ g_{x2} & g_{y2} \\ \vdots & \vdots \\ g_{xk} & g_{yk} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \tag{4.40}$$

which we write as

$$\underbrace{\mathbf{G}}_{k\times 2}\underbrace{\mathbf{u}}_{2\times 1}=\underbrace{\mathbf{B}}_{k\times 1} \tag{4.41}$$

This system can be solved for $k \geq 2$ in the least-square error sense as follows. First, we make the system square:

$$\mathbf{G}^{\top}\mathbf{G}\mathbf{u}=\mathbf{G}^{\top}\mathbf{B} \tag{4.42}$$

Second, we solve it in the least-square error sense:

$$\mathbf{u}=\left(\mathbf{G}^{\top}\mathbf{G}\right)^{-1}\mathbf{G}^{\top}\mathbf{B} \tag{4.43}$$

Done. $\mathbf{G}^{\top}\mathbf{G}$ is a 2×2 matrix, while $\mathbf{G}^{\top}\mathbf{B}$ is a 2×1 matrix. For example, if

$$\mathbf{G}^{\top}\mathbf{G}=\begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{4.44}$$

then

$$\left(\mathbf{G}^{\top}\mathbf{G}\right)^{-1}=\frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{4.45}$$

The rest is simple matrix multiplication.

Original Lucas-Kanade Optical flow Algorithm & Algorithm with weights

Original Lucas–Kanade Optical Flow Algorithm The basic algorithm is as follows:

1. Decide for a local neighbourhood of k pixels and apply this uniformly in each frame.
2. At frame t, estimate I_x, I_y, and I_t.
3. For each pixel location p in Frame t, obtain the equation system (4.40) and solve it in the least-squares sense as defined in (4.43).

It might be of benefit to estimate the used derivatives on smoothed images, for example with a Gaussian filter with a small standard deviation such as $\sigma = 1.5$.

Weights for Contributing Pixels We weight all the k contributing pixels by positive weights w_i for $i = 1, \ldots, k$. In general, the *current pixel* has the maximum weight, and all the adjacent pixels (contributing to those k pixels) have smaller weights. Let $\mathbf{W} = \text{diag}[w_1, \ldots, w_k]$ be the $k \times k$ diagonal matrix of those weights. A diagonal matrix satisfies

$$W^T W = WW = W^2$$

The task is now to solve the equation

$$WAu = WB \quad \text{— eq}^n 4.47$$

$$(WA)^T WAu = (WA)^T WB$$

$$A^T W^T WAu = A^T W^T WB$$

$$A^T W^2 Au = A^T W^2 B$$

$$\text{Thus } u = [A^T W^2 A]^{-1} A^T W^2 B$$

The meaning of those transforms is that we again have a 2×2 matrix $\mathbf{G}^\top \mathbf{W}^2 \mathbf{G}$, for which we can use the inverse for calculating $\mathbf{u}$ at the current pixel.

Compared to the original algorithm, we only need to change Step 3 into the following:

> 3. For each pixel location p in Frame t, obtain the equation system (4.47) and solve it in the least-squares sense as defined in (4.52).

Figure 4.15 illustrates results for the original Lucas–Kanade algorithm for a 5×5 neighbourhood. When solving by using (4.43), we only accept solutions for cases where the eigenvalues (see Insert 2.9) of the matrix $\mathbf{G}^\top \mathbf{G}$ are greater than a chosen threshold. By doing so we filter out "noisy" results.

The matrix $\mathbf{G}^T \mathbf{G}$ is a 2×2 (symmetric positive definite) matrix; it has two eigenvalues λ_1 and λ_2. Let $0 \leq \lambda_1 \leq \lambda_2$. Assume a threshold $T > 0$. We "accept" the solution provided by (4.43) if $\lambda_1 \geq T$. This leads to a sparse optical flow field as illustrated in Fig. 4.15. Note that those few shown vectors appear to be close to the true displacement in general.

The BBPW Algorithm

Advantages of Horn Schunk Algorithm

1) Improved result in case of large displacement

2) Overcome limitations defined by ICA

3) Use of square in optimization approach allows outliers to have significant impact; thus an L_1 optimization approach is used instead of L_2

approach of Horn Schunk Algorithm

The intensity constancy assumption (ICA) is in the considered context formally represented by $I(x, y, t) = I(x + u, y + v, t + 1)$, with u and v being the translations in x- and y-directions, respectively, during a δt time interval. Linearized, this became the *Horn–Schunck constraint* $I_x u + I_y u + I_t = 0$. Intensity constancy is not true for outdoor scenarios; as a result, Horn–Schunck or Lucas–Kanade algorithms often work poorly for pixel changes over, say, five to ten pixels (i.e. for 2D motion vectors that are not "short" in the sense of the only used linear term of the Taylor expansion).

Gradient Constancy The assumption of *constancy of intensity gradients over displacements* (GCA) is represented as

$$\nabla_{x,y} I(x, y, t) = \nabla_{x,y} I(x + u, y + v, t + 1) \tag{4.53}$$

where $\nabla_{x,y}$ is limited, as before, to the derivatives in the x- and y-directions only; it does not include the temporal derivative. As already discussed in Chap. 1, gradient information is considered to be fairly robust against intensity changes.

Smoothness Assumption Using only ICA and GCA does not provide sufficient information for having optical flow uniquely identified; we need to involve adjacent pixels and also to introduce some consistency into the calculated flow vector field.

Care needs to be taken at object boundaries. Motion discontinuities should not be fully excluded; piecewise smoothing appears as an option for avoiding the flow vectors from outside an object affect the flow field within the object, and vice versa.

A first Draft of an Energy Formula In the following draft of the error or energy function we still follow the ICA (but without going to the Horn–Schunck constraint, thus also avoiding the approximation in the Taylor expansion), but combined with the GCA. We consider pixel locations (x, y, t) in the frame $I(\cdot, \cdot, t)$, optical flow vectors $\mathbf{w} = (u, v, 1)^\top$, which also contain the third component for going from an image plane at time t to the plane at time $t + 1$, and use the gradient $\nabla = (\partial_x, \partial_y, \partial_t)$ in the 3D space. Equation (4.9) turns now into

$$\begin{aligned} E_{data}(f) = \sum_{\Omega} \big(& \big[I(x + u, y + v, t + 1) - I(x, y, t) \big]^2 \\ & + \lambda_1 \cdot \big[\nabla_{x,y} I(x + u, y + v, t + 1) - \nabla_{x,y} I(x, y, t) \big]^2 \big) \end{aligned} \tag{4.54}$$

where $\lambda_1 > 0$ is some weight to be specified. For the smoothness term, we basically stay with (4.10), but now for spatio-temporal gradients in the 3D space and in the formulation

$$E_{smooth}(f) = \sum_{\Omega} \big[\|\nabla u\|_2^2 + \|\nabla v\|_2^2 \big] \tag{4.55}$$

Altogether, the task would be to identify a labelling function f that assigns optical flow u and v to each pixel in $I(\cdot, \cdot, t)$ such that the sum

$$E_{total}(f) = E_{data}(f) + \lambda_2 \cdot E_{smooth}(f) \tag{4.56}$$

is minimized for some weight $\lambda_2 > 0$. As in (4.11), (4.56) still uses quadratic penalizers in the L_2 sense, thus defining a TVL_2 optimization problem. Outliers are given too much weight in this scheme for estimating optic flow.

Regularized L_1 Total Variation A move away from L_2-optimization towards L_1-optimization is by using the function

$$\Psi\left(s^2\right) = \sqrt{s^2 + \varepsilon} \approx |s| \tag{4.57}$$

rather than $|s|$ itself in the energy function. This leads to a more robust energy function where we still may consider continuous derivatives at $s = 0$, even for a small positive constant ε. The constant ε can be, for example, equal 10^{-6} (i.e., a very small value). There are no continuous derivatives for $|s|$ at $s = 0$, but we need those for applying an error-minimization scheme.

The function $\Psi(s^2) = \sqrt{s^2 + \varepsilon}$, an increasing concave function, is applied over the error-function terms in (4.54) and (4.55) to reduce the influence of outliers. This defines a *regularized total-variation term* in the L_1 sense. We obtain the energy terms

$$E_{data}(f) = \sum_{\Omega} \Psi\Big(\big[I(x+u, y+v, t+1) - I(x, y, t)\big]^2 + \lambda_1 \cdot \big[\nabla_{x,y} I(x+u, y+v, t+1) - \nabla_{x,y} I(x, y, t)\big]^2\Big) \quad (4.58)$$

and

$$E_{smooth}(f) = \sum_{\Omega} \Psi\big((\nabla u)^2 + (\nabla v)^2\big) \quad (4.59)$$

The way for obtaining the total energy remains as specified by (4.56).

Outline of the algorithm

We have to find a labelling f that minimizes the error function E_{total} as defined in (4.56), using the error terms of (4.58) and (4.59).

Minimizing a nonlinear function such as E_{total} is a difficult problem. When searching for the global minimum, a minimization algorithm could become trapped in a local minimum.

Euler Lagrange Equations

Euler–Lagrange Equations Let Ψ' be the derivative of Ψ with respect to its only argument. The *divergence* div denotes the sum of derivatives, in our case below the sum of three partial derivatives, being the components of

$$\nabla u = \left[\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial t}\right]^\top \quad (4.60)$$

A minimizing labelling function f for the functional $E_{total}(f)$ has to satisfy the following *Euler–Lagrange equations*. We do not provide details about the derivation of those equations using total-variation calculus and just state the resulting equations:

$$\Psi'\left(I_t^2 + \lambda_1\left(I_{xt}^2 + I_{yt}^2\right)\right) \cdot \left(I_x I_t + \lambda_1 (I_{xx} I_{xt} + I_{xy} I_{yt})\right) - \lambda_2 \cdot \operatorname{div}\left(\Psi'\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)\nabla u\right) = 0 \tag{4.61}$$

$$\Psi'\left(I_t^2 + \lambda_1\left(I_{xt}^2 + I_{yt}^2\right)\right) \cdot \left(I_y I_t + \lambda_1 (I_{yy} I_{yt} + I_{xy} I_{xt})\right) - \lambda_2 \cdot \operatorname{div}\left(\Psi'\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)\nabla v\right) = 0 \tag{4.62}$$

As before, I_x, I_y, and I_t are the derivatives of $I(\cdot, \cdot, t)$ with respect to pixel coordinates or time, and I_{xx}, I_{xy}, and so forth are 2nd-order derivatives. All those derivatives can be considered to be constants, to be calculated based on approximations in the sequence of frames. Thus, both equations are of the form

$$c_1 - \lambda_2 \cdot \operatorname{div}\left(\Psi'\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)\nabla u\right) = 0 \tag{4.63}$$

$$c_2 - \lambda_2 \cdot \operatorname{div}\left(\Psi'\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)\nabla v\right) = 0 \tag{4.64}$$

The solution of these equations for u and v at any pixel $p \in \Omega$ can be supported by using a pyramidal approach.

Pyramidal Algorithm It is efficient to use down-sampled copies of the processed frames in an image pyramid (see Sect. 2.2.2) for estimating flow vectors first and then use the results in higher-resolution copies for further refinement. This also supports the identification of long flow vectors. See Fig. 4.16 for an example for a pyramidal implementation of the BBPW algorithm. We use the same colour key as introduced by Fig. 4.4.

4.5 Performance Evaluation of Optical Flow Results

The Horn–Schunck algorithm was a pioneering proposal for calculating optical flow; many other algorithms have been designed till today. Motion analysis is still a very challenging subject in computer vision research.

This section informs briefly about (comparative) performance evaluation techniques for optical flows.

4.5.1 Test Strategies

The Horn–Schunck algorithm is in general moving away from a gradient flow. The following simple example (which allows even manual verification) shows that this is not happening if pixel neighbourhoods do not allow computation of a correct motion.

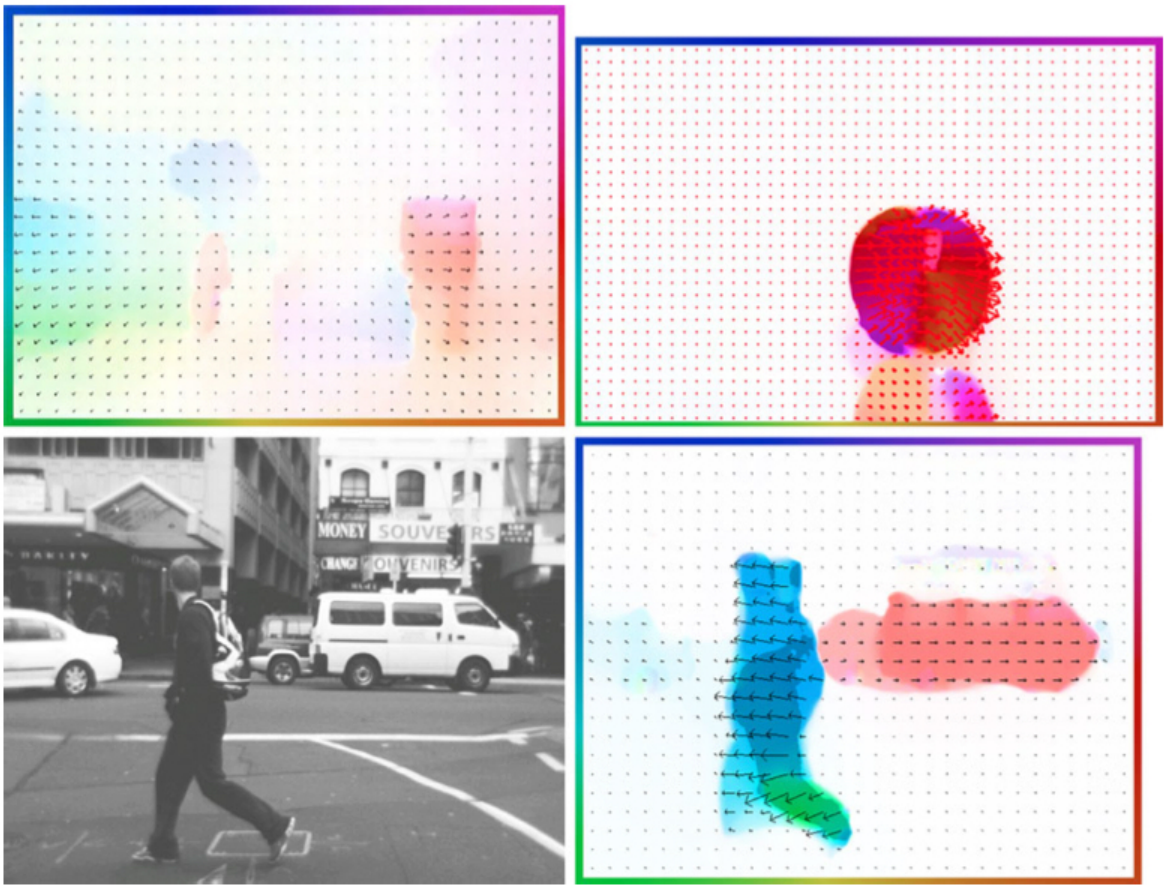

Fig. 4.16 Visualization of calculated optical flow using a pyramidal BBPW algorithm. *Top*: For frames shown in Fig. 4.5. *Bottom*: For a frame in the sequence `queenStreet`

Example 4.2 (A Simple Input Example) Assume a 16×16 image I_1 as shown in Fig. 4.17, with $G_{\max} = 7$. It contains a vertical linear edge; the gradient at edge pixels points to the left.

The image I_2 is generated by one of the three motions sketched on the left: (A) is a one-pixel shift (0,1) to the right, (B) is a diagonal shift (1,1), and (C) a one-pixel shift $(-1, 0)$ upward. For simplicity, we assume that additional values (i.e. which are "moving in") on the left are zero and at the bottom identical to the given column values.

This simple example allows you to calculate manually values as produced by the Horn–Schunck algorithm, say in the first three iterations only [(4.22) and (4.22)], using zero as initialization, and simple two-pixel approximation schemes for I_x, I_y, and I_t.

Performance Evaluation of Optic Flow Algorithms For evaluating an optical flow algorithm on real-world image sequences, there are different options, such as the following:

1. Assuming that we have high-speed image sequence recording (say, with more than 100 Hz), it might be possible to use alternating frames for *prediction-error analysis*: estimate optical flow for Frames t and $t+2$, calculate a virtual image for

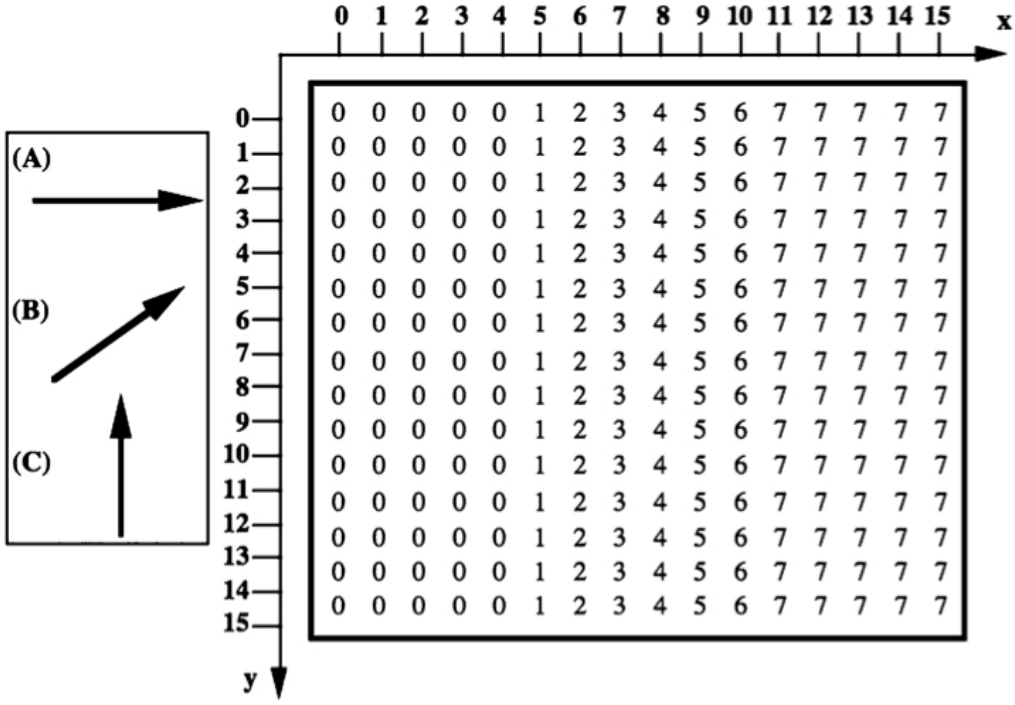

Fig. 4.17 A simple input for a manual experience of the Horn–Schunck iteration

$t + 1$ by interpolating along the calculated flow vectors, and compare the virtual image (e.g. use *normalized cross-correlation*; see insert) with Frame $t + 1$.

2. If there is ground truth available about the actual 2D motion (e.g. such as on the website vision.middlebury.edu/ at Middlebury College for short sequences, or in Set 2 on EISATS for sequences of 100 frames or more), the evaluation of optical flow results may be based on *error measures* (see below) by comparing true (well, modulo measurement errors when generating the ground truth) and estimated vectors.
3. There is also the option to compare results of multiple methods, for understanding which methods generate similar results and which methods differ in their results.
4. A method can be evaluated on real-world data by introducing incrementally different degrees of various kinds of noise into the input data for understanding robustness in results. Noise might be, for example, intensity variations, Gaussian noise, or blurring.
5. Known scene geometry and approximately known camera motion can be used to compare the results with expected flow fields. For example, the camera may move steadily towards a wall.

Insert 4.11 (Normalized Cross-Correlation) *Assume two images I and J of identical size. The normalized cross-correlation (NCC) compares both images and provides one scalar response*:

$$M_{NCC}(I,J) = \frac{1}{|\Omega|} \sum_{(x,y)\in\Omega} \frac{[I(x,y) - \mu_I][J(x,y) - \mu_J]}{\sigma_I \sigma_J}$$

μ_I and μ_J denote the means, and σ_I and σ_J the standard deviations of I and J, respectively. The larger $M_{NCC}(I,J)$, the more similar both images are.

Due to normalizing with respect to mean and variance, the images I and J may differ in these two values and can be still very similar. This is in particular of interest when comparing images that do not follow the ICA, which can happen for two subsequently recorded frames in an image sequence.

The NCC can also be used to compare a small image (the template) with image windows of the same size of the template in a larger image. `OpenCV` *provides several methods for this kind of template matching, also including variants of normalized cross-correlation.*

4.5.2 Error Measures for Available Ground Truth

Ground truth for motion analysis is difficult to obtain, especially for real-world sequences. Synthesized sequences are an alternative option. Physics-based image rendering can lead to "fairly realistic" synthetic image sequences, even with an option to study behaviour of optical flow algorithms for varying parameters (something that cannot be done for recorded real-world sequences).

Insert 4.12 (Ground Truth) *Images recorded in an airplane are often used to estimate distances on the ground, or even for 3D reconstruction of whole landscapes or cities. For evaluating results, it was common practice to identify landmarks on the ground, such as corners of buildings, and to measure distances or positions of those landmarks. This was the ground truth, to be compared with the values calculated based on the images recorded in an airplane.*

The term is now in general use for denoting measured data, considered to be fairly accurate, thus useful for evaluating algorithms supposed to provide the same data.

Figure 4.18 illustrates provided ground truth for a synthetic image sequence (of 100 frames) and results of a pyramidal Horn–Schunck algorithm. Additionally to the used colour key, we also show sparse vectors in the ground-truth image and result image. This is redundant information (and similar to a needle map, as shown in Fig. 4.12) but helps a viewer to understand occurring motions. Also, the coloured frame around the visualized flow vectors shows colour values corresponding to all the possible directions.

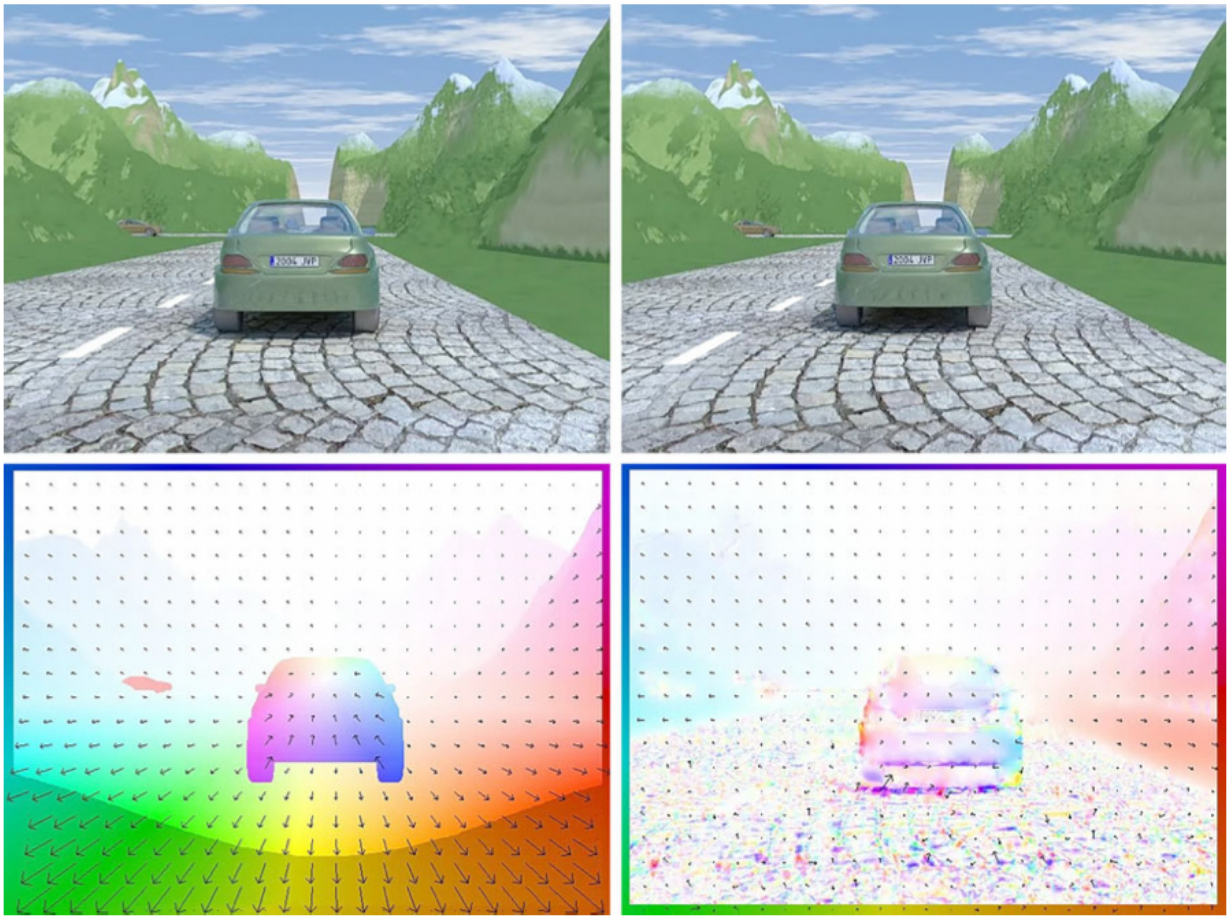

Fig. 4.18 *Top*: Two input images (Frame 1, `Set2Seq1`, and Frame 2 from the first sequence of Set 2 of EISATS). *Bottom, left*: Ground truth as provided for this sequence, using a colour key for visualizing local displacements. *Bottom, right*: Result of a pyramidal Horn–Schunck algorithm

Error Measures in the Presence of Ground Truth We have the calculated flow $\mathbf{u} = (u, v)$ and flow $\mathbf{u}^\star = (u^\star, v^\star)$ provided as a ground truth. The *L_2 endpoint error*

$$E_{ep2}(\mathbf{u}, \mathbf{u}^\star) = \sqrt{(u - u^\star)^2 + (v - v^\star)^2}$$

or the *L_1 endpoint error*

$$E_{ep1}(\mathbf{u}, \mathbf{u}^\star) = |u - u^\star| + |v - v^\star|$$

compare both vectors in 2D space.

The *angular error* between spatio-temporal directions of the estimated flow and ground truth at a pixel location p (not included in the formula) is as follows:

$$E_{ang}(\mathbf{u}, \mathbf{u}^\star) = \arccos\left(\frac{\mathbf{u}^\top \cdot \mathbf{u}^\star}{|\mathbf{u}|\ |\mathbf{u}^\star|}\right) \tag{4.65}$$

where $\mathbf{u} = (u, v, 1)$ is the estimated optical flow, but now extended by one coordinate (the 1 corresponds to the distance between images recorded at time slots t and $t + 1$) into the 3D space, and $\mathbf{u}^\star = (u_t, v_t, 1)$ is the flow provided as a ground

truth, also extended into the 3D space. This error measure evaluates accuracy in both direction and magnitude in the 3D space.

Such error measures are applied to all pixels in a frame, which allows us to calculate the *mean angular error*, the *mean* L_2 *endpoint error*, and so forth, also with their standard deviations.