CHAPTER 35

Estimating Optic Flow

To estimate flow, you must determine where a pixel in image 1 went to in image 2. Photometric consistency implies that the correct pixel in image 2 must have the same color as the pixel in image 1. This means you are interested in flow fields $\mathbf{v}$ that have a small value of

$$\mathcal{L}_{\text{photo}}(\mathcal{I}(\mathbf{x}+\mathbf{v},t),\mathcal{I}(\mathbf{x},t))$$

where $\mathcal{L}_{\text{photo}}$ is a photometric loss that compares its two arguments in some way.

There are many $\mathbf{v}$ that will minimize the loss. Estimation involves choosing something that (a) makes the loss small and (b) is a reasonable flow field. Photometric consistency is hard to work with, because $\mathcal{I}(\mathbf{x}+\mathbf{v},t)$ isn't linear in $\mathbf{v}$. You can linearize photometric consistency to manage this issue (Section 35.1.5). Linearizing creates new problems, which can be controlled with coarse-to-fine estimation and iterative re-estimation (Section 35.2.3). Photometric consistency is not enough because there will be many pixels in image 2 with the right color; and in some situations, photometric consistency doesn't apply. You can manage these issues by evaluating photometric consistency over patches of an image, rather than pixels (Section 35.2.2). The collection of possible $\mathbf{v}$ is extremely large (one vector per pixel). You can manage this issue by forcing the flow field to be smooth; by computing photometric consistency over patches, which is a form of smoothing; or by working with parametric flow models.

35.1 ESTIMATING FLOW FIELDS: OUTLINE

Procedures for estimating flow apply a version of the master recipe for denoising: find a flow field that is (a) close to consistent with image data and (b) "like" a real flow field.

Remember this: *The master recipe for estimating flow balances the extent to which flow is consistent with data against the extent to which the flow field is "like" a real flow field. The recipe uses two cost functions. Write λ for a weight that balances the two terms. Choose $\mathbf{u}$ that minimizes*

$$\begin{aligned} C(\mathbf{u}) &= \begin{array}{c}[\textit{the extent to which } \mathbf{u} \textit{ is consistent with image data}] + \\ \lambda\,[\textit{the extent to which } \mathbf{u} \textit{ is like a real flow field}]\end{array} \\ &= [\text{data term}] + \lambda\,[\text{penalty term}]\,. \end{aligned}$$

The penalty term is essential. Photometric consistency is much more ambiguous than you might think, and comes with various annoying caveats. Forcing the

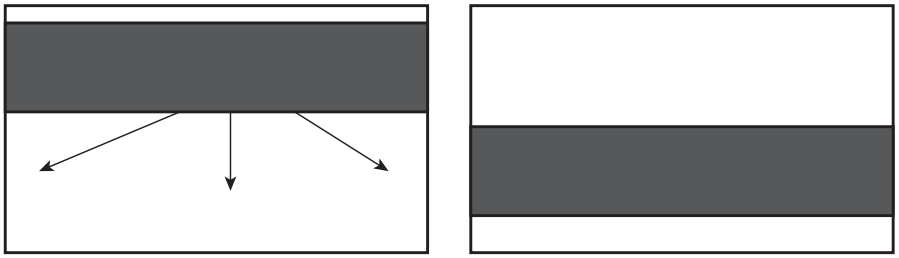

FIGURE 35.1: *Each of the three flow vectors is a plausible explanation of the flow on this moving bar, because you cannot tell how it has translated horizontally. This effect is known as the aperture problem.*

flow field to be "like" a real flow field is an important method to control ambiguity. However, real flow fields aren't smooth, so forcing a flow field to be "like" a real flow field by just forcing it to be smooth is dangerous. Worse, in a real flow field there are some locations where flow is indeterminate – places where you can see something in the first image that you can't see in the second image, for example.

35.1.1 Photometric Consistency is Ambiguous

The most ambiguous flow occurs when a pixel is inside a region of constant color – the pixel can go to any one of many different pixels in the region in the second image. In the same way, the flow at a straight edge is ambiguous, because you can't tell whether there is any flow *along* the edge wherever the edge is straight. The flow is constrained because you know the component across the edge, but you don't know it uniquely. Figure 35.1 illustrates an exaggerated version of this problem. This isn't just a property of edges. If the line segment in the figure was an *isophote* (a curve of equal brightness points in the image) the same problem would occur. You would not be able to tell *locally* if there was any flow *along* the isophote. This problem is known as the *aperture problem*.

Remember this: *At most pixels, the flow estimate is ambiguous. In constant regions, flow is indeterminate; along edges or isophotes, the flow is subject to the aperture problem – you can determine the component perpendicular to the edge (or isophote), but not the tangential component.*

Flow at corners is unambiguous. This shouldn't come as a surprise. You can tell where corners are, so you can tell where they have gone to. This is the keystone of flow estimation. Any flow estimation procedure has to "fill in" flow between the scattered locations where it is unambiguous. The "filled in" flow needs to meet constraints at edges, but must involve some kind of smoothing.

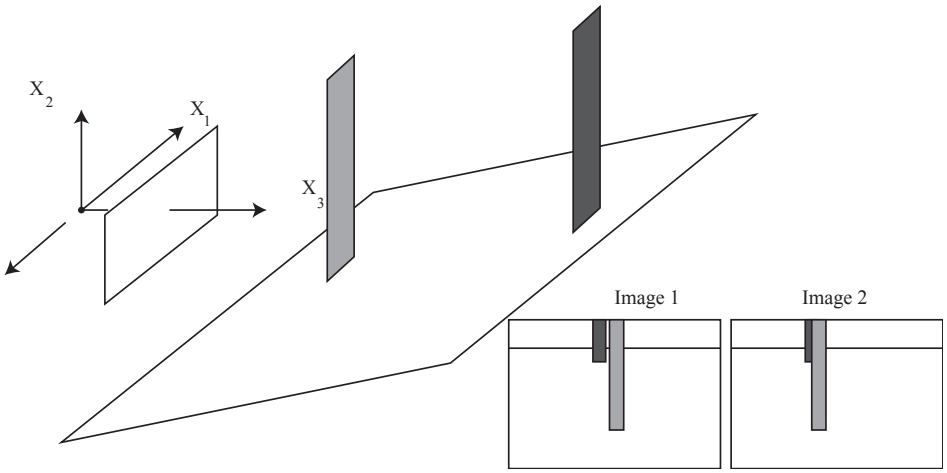

FIGURE 35.2: *A camera looks out of a train window at right angles to the direction of travel (the small arrow) and views a ground plane with two trees on it. Because the nearby tree moves by more than the distant tree (as in Figure 34.10), for this geometry the distant tree will disappear behind the nearby tree. Reversing the direction of camera movement will get an example where the distant tree reappears from behind the nearby tree. In either case, there are pixels in one image that aren't seen in the other, which must create estimation problems.*

35.1.2 Flow Fields aren't Smooth

But mostly, flow fields aren't smooth. For example, Figure 34.10 shows the flow field when there are two trees sticking out of the ground plane. The flow inside the boundary of either tree is very different from that outside the boundary. From Section 34.1 and this example it follows that you should expect discontinuities in flow at discontinuities in depth. These discontinuities in flow should – and do – generate problems estimating the flow.

> **Remember this:** *Flow at corners isn't ambiguous. Flow estimation must involve some process of "filling in" flow between the scattered locations where flow is known. Just smoothing may not be enough, because flow fields are not usually smooth.*

35.1.3 Flow is not defined Everywhere

In most flow fields, there are pixels where flow isn't meaningful. You can't estimate flow at a pixel that is there in one image but not in the other image. This occurs at *occlusions* – something visible in the first image is hidden behind something else in the second image. Parts of the distant tree disappear behind the nearby tree in

FIGURE 35.3: *Fast moving objects produce heavily blurred images. These are five consecutive frames of a sequence where I dropped a pen on my laptop. Notice in the* **far left** *frame, my hand is in focus as is the pen. Dropping the pen involves a fairly fast movement of my thumb, blurred in the* **left** *frame. As the pen accelerates under gravity, it is increasingly blurred.*

Figure **??**, which must create estimation problems. Similarly, parts of the distant tree might reappear from behind the nearby tree.

Remember this: *At some pixels, flow is indeterminate, because the point that produced the pixel isn't visible in both images.*

35.1.4 Motion Blur affects Flow

Definition: 35.1 *Motion blur*

A fast moving object may cover a pixel only for part of the time that the camera shutter is open. The value reported by the pixel is some weighted average of the background color and the object color. Some pixels may see mostly background, and others will see mostly object. This produces *motion blur* around the boundary of the object, which can be quite pronounced.

Motion blur (Figure 35.3) can significantly affect flow estimates. Motion blur will affect the image gradient estimates and the time derivative of intensity, and so tend to make the flow constraint unreliable. Small, fast moving objects are particularly bad, because there may be few pixels on the object and all could be affected by motion blur. Further, the definition of flow becomes problematic, because a pixel in image 1 may actually be mapped to several pixels in image 2. Worse, rather than one pixel in image 1 corresponding to one pixel in image 2, the "true" flow might involve bits of several pixels in image 1 being shared among several pixels in image 2.

35.1.5 Linearizing Photometric Consistency

Measure photometric consistency using intensity. Obtain images at time t and time $t+\delta t$. In the ideal case, if $\mathbf{x}$ in the first image corresponds to $\mathbf{x}+(\delta t)\mathbf{u}$ in the second

image, you expect these points to have the same intensity, yielding

$$I(\mathbf{x} + (\delta t)\mathbf{u}, t + \delta t) - I(\mathbf{x}, t) = 0.$$

If the flow and δt are small enough, a Taylor series is a good approximation, and you find

$$I(\mathbf{x} + (\delta t)\mathbf{u}, t + \delta t) \approx (I(\mathbf{x}, t) + (\delta t)\left[(\nabla I)^T\mathbf{u} + \frac{\partial I}{\partial t}\right]$$

so that for sufficiently small δt

$$(\nabla I)^T\mathbf{u} + \frac{\partial I}{\partial t} = 0.$$

This linearization is known as the *optic flow equation*.

Definition: 35.2 *The optic flow equation*

Assume sufficiently small flow field $\mathbf{u}$, obtained over a sufficiently small timestep. A Taylor series yields

$$(\nabla I)^T\mathbf{u} + \frac{\partial I}{\partial t} = 0.$$

which is known as the optic flow equation.

At any pixel, it is easy to compute estimates ∇I and $\frac{\partial I}{\partial t}$, so this equation could yield one linear constraint on the flow vector $\mathbf{u}$. This isn't enough to recover the flow vector exactly, but can strongly constrain the flow. The linearized constraint has some issues. The movement might be so large that linearizing the expression is reckless, and knowing when this occurs can be difficult. If the image has many sharp edges, the linearized constraint can break down quite quickly.

35.1.6 Horn-Schunk Flow Estimation

Here is the simplest instance of the master recipe. Assume the linearized version of the photometric consistency constraint applies. You do not know the image gradient or the time derivative exactly, but you can estimate them. There will be small errors in the derivatives, so you do not expect the optic flow equation to vanish exactly. Instead, use

$$D(u, v) = \sum_{i,j \in \text{pixels}} \left((\nabla I_{ij})^T\mathbf{u}_{ij} + \frac{\partial I_{ij}}{\partial t}\right)^2$$

as a data term in the master recipe. Here the subscripts refer to the value at the locations (so ∇I_{ij} is the gradient of I at i, j and so on). If this term is small, the optic flow equation is "mostly true".

Now assume that the gradient of each flow component is small. This gives a penalty term. Flow components are u, v, so

$$P(u,v) = \sum_{i,j \in \text{pixels}} \left([\nabla u_{ij}]^T [\nabla u_{ij}] + [\nabla v_{ij}]^T [\nabla v_{ij}] \right)$$

should be small. Then $C(u,v) = D(u,v) + \lambda P(u,v)$ where λ weights the penalty term against the data term. This cost function is quadratic in u and v, so a solution is obtained with linear algebra. Solving the resulting flow equation is straightforward, but you should expect reasonable results only for very small flows (**exercises**). The method I have described is known as the *Horn-Schunk* method, and was originally described in []. It is now mostly obsolete.

35.2 ESTIMATING FLOW USING LOCAL IMAGE WINDOWS

The ambiguity of Section 35.1 means you are forced to assume that flow is smooth in some way. If you assume that the flow in a small image window is constant, you could estimate the flow by matching windows to windows. One way to do this is to build a simple model of the variation of the intensity around a point, then reason about how flow changes that model.

35.2.1 Local Quadratic Models of Intensity

The simplest procedure fits a quadratic model to the image window around each point. At each position $\mathbf{x}$ in the image, compute $\mathcal{A}(\mathbf{x})$, $\mathbf{b}(\mathbf{x})$ and $c(\mathbf{x})$ that give a least-squares fit to the image intensity in the $k \times k$ window centered at $\mathbf{x}$. Write $\mathbf{x} = (i,j)^T$, I_{ij} for $\mathcal{I}(\mathbf{x})$. The matrix is symmetric, so write

$$\mathcal{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}.$$

The model for intensity at $i+u, j+v$ is then

$$a_{11}u^2 + 2a_{12}uv + a_{22}v^2 + b_1 u + b_2 v + c.$$

This model summarizes the intensities in a window about the point. Its elements are analogous to the elements of a Taylor series.

Procedure: 35.1 *Fitting a quadratic model to a $(2k+1)\times(2k+1)$ image window*

Use the notation above. Write $\mathbf{a} = (a_{11}, a_{12}, a_2, b_1, b_2, c)^T$,

$$\mathbf{d} = \begin{bmatrix} I_{i-k,j-k} \\ I_{i-k,j-k+1} \\ \dots \\ I_{i+k,j+k-1} \\ I_{i+k,j+k} \end{bmatrix}$$

and

$$\mathcal{M} = \begin{bmatrix} k^2 & k^2 & k^2 & -k & -k & 1 \\ k^2 & k(k-1) & (k-1)^2 & -k & -(k-1) & 1 \\ \dots & & & & & \dots \\ k^2 & k(k-1) & (k-1)^2 & k & (k-1) & 1 \\ k^2 & k^2 & k^2 & k & k & 1 \end{bmatrix}.$$

Obtain $\mathbf{a}$ by solving

$$\mathcal{M}^T\mathcal{M}\mathbf{a} = \mathcal{M}^T\mathbf{d}$$

Notice that you will solve a linear system where for each $\mathbf{x}$, the matrix is the same. This means you could speed things up by forming the matrix, constructing a Cholesky or LU decomposition, and reusing that to solve the linear system at each point.

exercises

35.2.2 Estimating Flow with a Local Quadratic Model

The window at $\mathbf{x}_1$ in $\mathcal{I}_1$ is represented by $\mathcal{A}_1(\mathbf{x}_1)$, $\mathbf{b}_1(\mathbf{x}_1)$ and $c_1(\mathbf{x}_1)$. Assume the flow translates that window by $\mathbf{t}$, and think about the window at $\mathbf{x}_2' = (\mathbf{x}_1 - \mathbf{t})$ in $\mathcal{I}_2$. You will have $\mathcal{A}_2(\mathbf{x}_2') = \mathcal{A}_1(\mathbf{x}_1)$, etc. because the flow has translated the image. This is not much help, because you don't know the translation, so you don't know $\mathbf{x}_2'$. But if the flow is small, you will be able to determine it from $\mathcal{A}_2(\mathbf{x}_1)$ and $\mathbf{b}_2(\mathbf{x}_1)$, by the following argument.

If $\mathcal{A}_1(\mathbf{x}_1)$, $\mathbf{b}_1(\mathbf{x}_1)$ and $c_1(\mathbf{x}_1)$ are an accurate local representation of $\mathcal{I}_1(\mathbf{x}_1)$, if $\mathcal{I}_2(\mathbf{x}) = \mathcal{I}_1(\mathbf{x} - \mathbf{t})$, and if $\mathbf{t}$ is sufficiently small, then

$$\mathcal{A}_2 = \mathcal{A}_1,\ \mathbf{b}_2 = \mathbf{b}_1 - 2\mathcal{A}_1\mathbf{t}, \text{ and } c_2 = c - \mathbf{b}_1^T\mathbf{t} + \mathbf{t}^T\mathcal{A}_1\mathbf{t}$$

are an accurate representation of $\mathcal{I}_2(\mathbf{x}_1)$. You get this result by substituting expressions for $\mathbf{t}$ into the fitting equation **exercises**. Ignore the effect of flow on the constant. It is inconvenient to work with and is also not a particularly reliable cue **exercises**. With some rearrangement, you get a flow estimate.

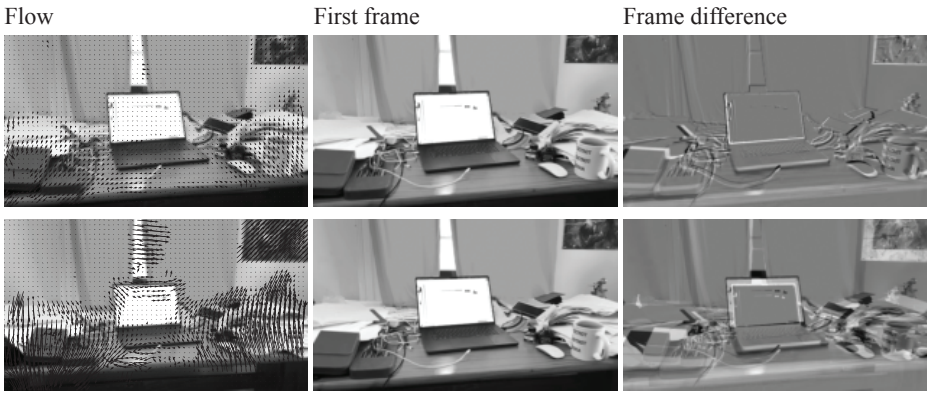

FIGURE 35.4: *Estimates of flow for a pair of images of my laptop, obtained with the method of Section* **??**. **Top** *row shows estimates obtained with a small elapsed time;* **bottom** *shows estimates obtained with a large elapsed time.* **Left** *is the first frame with flow superimposed as small arrows;* **center** *is the second frame; and* **right** *is second frame - first frame, scaled so that zero is gray, negative numbers are dark, and positive numbers are light. This makes it possible for you to check the flow estimates – if, reading left to right, you see a vertical light bar closely followed by a vertical dark bar at the left edge of a bright region, the second frame is moving left.*

Procedure: 35.2 *Estimating flow from a quadratic model*

Choose a window size k. Form $\mathcal{A}_1(\mathbf{x})$, $\mathcal{A}_2(\mathbf{x})$, $\mathbf{b}_1(\mathbf{x})$, and $\mathbf{b}_2(\mathbf{x})$. Then, if the flow at $\mathbf{x}$ is sufficiently small,

$$(\mathcal{A}_1(\mathbf{x}) + \mathcal{A}_2(\mathbf{x}))\,\mathbf{u}(\mathbf{x}) = \mathbf{b}_2(\mathbf{x}) - \mathbf{b}_1(\mathbf{x})$$

yields an estimate of the flow $\mathbf{u}(\mathbf{x})$.

exercises

This estimate involves solving a small linear system at each pixel location
exercises. You can smooth the estimate without engaging in large-scale linear algebra by forming local weighted estimates of the matrices.

Procedure: 35.3 *Smoothed flow from a quadratic model*

Choose a window size k. Choose a smoothing window, which will be $(2r+1) \times (2r+1)$. Choose a set of weights w_{ij} for this window. Form

$$\mathcal{C}_1(\mathbf{x}) = \sum_{u=-r}^{u=r} \sum_{v=-r}^{v=r} w_{uv} \left(\mathcal{A}_1(\mathbf{x} + (u,v)^T) + \mathcal{A}_2(\mathbf{x} + (u,v)^T) \right)$$

and

$$\mathbf{d}_1(\mathbf{x}) = \sum_{u=-r}^{u=r} \sum_{v=-r}^{v=r} w_{uv} \left(\mathbf{b}_2(\mathbf{x} + (u,v)^T) - \mathbf{b}_1(\mathbf{x} + (u,v)^T) \right)$$

Then, if the flow at $\mathbf{x}$ is sufficiently small,

$$\mathcal{C}_1 \mathbf{u}(\mathbf{x}) = \mathbf{d}_1(\mathbf{x})$$

yields an estimate of the flow $\mathbf{u}(\mathbf{x})$.

exercises Typically, one uses gaussian weights. This method is often referred to as the *Farneback method*, and originates in []. Example flow estimates appear in Figure 35.4. Alternatively, you could smooth the flow estimate using the master recipe of Section **??**, as below.

Procedure: 35.4 *Globally smoothed flow*

Use the notation of Procedure 35.3. Choose a weight λ. Now obtain the flow $\mathbf{u}(\mathbf{x})$ that minimizes the cost function

$$\sum_{\mathbf{x} \in \text{pixels}} \left[\begin{array}{c} [\mathcal{C}_1 \mathbf{u}(\mathbf{x}) - \mathbf{d}_1(\mathbf{x})]^T [\mathcal{C}_1 \mathbf{u}(\mathbf{x}) - \mathbf{d}_1(\mathbf{x})] + \\ \lambda \left([\nabla u_1(\mathbf{x})]^T [\nabla u_1(\mathbf{x})] + [\nabla u_2(\mathbf{x})]^T [\nabla u_2(\mathbf{x})] \right) \end{array} \right]$$

yields an estimate of the flow $\mathbf{u}(\mathbf{x})$.

exercises

You should see the flow estimators above as solving a fitting problem, and so you should suspect there might be robustness issues. There are: the cure – or a useful emollient – is to replace the quadratic costs in various terms with a robust estimator. You would then apply IRLS (Section 21.3). Doing so is an instructive, if elaborate, exercise. I do not do this in detail because these flow estimators are now largely obsolete.

35.2.3 Coarse-to-Fine and Iterative Estimation

Approximating the image can cause serious difficulties and deserves more detailed attention. Both the linearized photometric constraint and the approximation used in Section **??** approximate $\mathcal{I}(\mathbf{x}+\mathbf{u})$. Neither approximation applies over a sufficiently long spatial scale, and it is quite hard to determine what is too long a scale. If the flow is too large, the approximation might fail and you will get dubious or meaningless flow vectors. It could be quite difficult to tell that the approximation had failed by just looking at flow vectors.

In *iterative estimation*, you assume that you have a fair estimate of the flow – write $\mathbf{u}^n$ – and want to get a better estimate $\mathbf{u}^{n+1} = \mathbf{u}^n + \delta\mathbf{u}$. You can assume that the update is small, and the linearized photometric constraint is

$$I(\mathbf{x}+\mathbf{u}^{n+1}, t+\delta t) \approx I(\mathbf{x}+\mathbf{u}^n) + (\nabla\mathcal{I})^T\delta\mathbf{u} + \frac{\partial\mathcal{I}}{\partial t}\delta t.$$

You then start with $\mathbf{u} = \mathbf{0}$, and repeatedly re-estimate until $\delta\mathbf{u}$ is very small or the estimates diverge. For this strategy to succeed, early estimates need to be good enough that small refinements are sufficient. This doesn't always happen.

In *coarse-to-fine estimation*, you notice that in quite smooth images, the gradient is a good guide to the appearance of the image quite a long way away from a pixel. In this case, you can expect the linearized constraint to be usable for quite large flows. But if the image is very smooth, there is no point in solving for flow at every pixel because flow at neighbors will be very like flow at the pixel. In turn, subsample the image, and solve for flow in the subsampled image. This should make you think about a gaussian pyramid.

Here is a starter recipe for a coarse-to-fine procedure. Form a gaussian pyramid for the two images. At the coarsest scale, estimate flow. Upsample this estimate to the next coarsest scale. Notice you have to (a) increase the number of locations to get a flow vector at every pixel, which you can do by interpolation and (b) increase the length of the flow vector, because moving by (say) a pixel at a coarse scale is equivalent to moving rather further at a less coarse scale. Now use the upsampled estimate as an initial estimate of flow, and linearize about that to estimate a correction. This is the iterative estimation recipe from above. Continue to upsample and correct until you arrive at the finest scale.

This is a starter recipe because there may be some advantage to passing the fine scale flow estimate down the pyramid to coarser scales. This option – which turns the flow estimator into a *multigrid method* is out of scope for us. You might start reading at [].

35.3 FLOW ESTIMATION WITH PARAMETRIC MODELS

The methods of Section 21.3 and Section 21.3 are actually parametric – you can see the $\mathbf{u}$ at each pixel as two parameters at each pixel – but they have a very large number of parameters and so are not typically thought of as parametric. The large number of parameters means the methods might be able to fit a wide range of flow fields quite accurately, but are more likely to make more errors. They control errors by smoothing the estimated flow field.

An alternative way to control errors is to use few parameters. If you *know* that the flow comes from a model like those of Section 34.1, then you need only estimate the parameters of the model. For example, if you know the flow is that of Section 34.2.2, you need to estimate $t_x b$ and $t_x c$. You will also need to estimate the horizon because the flow above the horizon will be zero. As the exercises show, $(t_x b)/(t_x c)$ yields the horizon.

35.3.1 Parametric Flow: A General Recipe

An important part of building good parametric flow estimators is coming up with parametric flow models that represent the flows you will encounter well with relatively few parameters. For the moment assume you have a good model.

Procedure: 35.5 *Parametric Flow Estimation*

Write $\mathbf{u}(\mathbf{x};\theta)$ for a parametric flow field, where θ are the parameters of the field. To obtain a flow field, you must choose some function $D(\cdot)$ for the data term to penalize the photometric inconsistency, another function – which you might even omit – $P(\theta)$ which penalizes the flow for being unrealistic. You then minimize

$$C(\theta) = \sum_{\mathbf{x} \in \text{pixels}} D(I_2(\mathbf{x} + \mathbf{u}(\mathbf{x};\theta)) - I_1(\mathbf{x})) + \alpha P(\theta)$$

as a function of θ.

In the simplest case you can linearize the photometric inconsistency, to get

$$\delta I(\mathbf{x};\theta) = (I_2(\mathbf{x}) - I_1(\mathbf{x})) - (\nabla I_1(\mathbf{x}))^T \mathbf{u}(\mathbf{x};\theta)$$

use $C(\delta I) = (\delta I)^2$ and set $\alpha = 0$. If the parametric flow model is linear in the parameters, you will then need to solve a linear system (**exercises**). You could do coarse-to-fine estimation with a robust cost function and IRLS, as sketched in Section 21.3. Because – by construction – you have relatively few parameters, you can use quite sophisticated optimization strategies like Newton's method or BFGS (see []). Some cases are explored in the **exercises**.

35.3.2 Estimating a Simple Mixture of Flows

Recall the simple layered motion model of Section 21.3. Assuming you don't know the ground plane constant, the translation, or the depth of any object, you still have only $N+1$ constants to estimate. Estimating these constants is tightly intertwined with segmenting the image into objects and ground plane. If the flow at pixel $\mathbf{x}$ is $\mathbf{u}$, then $I_2(\mathbf{x} + \mathbf{u})$ should be very similar to $I_1(\mathbf{x})$. The photometric error can be used to estimate this class of flow using an algorithm which is quite strongly analogous to k-means.

Procedure: 35.6 *Estimating a mixture of flows*

Iterate:

- estimate the flow assuming you know which object each pixel belongs to (Procedure 35.7);
- estimate which object a pixel belongs to using the flows (Procedure 35.8).

Procedure: 35.7 *Estimating flow from segmentation*

For every pixel, you know which of the $N+1$ cases applies (N objects, and background). For each case i, search for the $\mathbf{u}_i$ such that

$$\sum_{\mathbf{x} \in \text{pixels on } i} (I_2(\mathbf{x} + \mathbf{u}_i) - I_1(\mathbf{x}))^2$$

is minimized (for example, using methods of Section 21.3). Then $\mathbf{u}_i$ is the flow for that case.

Procedure: 35.8 *Estimating segmentation from flow*

You know the flow for each of the $N+1$ cases. For each pixel location $\mathbf{x}$, evaluate the photometric error produced by each flow $\mathbf{u}_i$. Allocate the segment that has the flow that produces the lowest photometric error, so the segment given by

$$\underset{i}{\text{argmin}} \; [\mathcal{I}_1(\mathbf{x} + \mathbf{u}_i) - \mathcal{I}(\mathbf{x})]^2$$

Notice how this approach links flow and segmentation. Pixels are collected together into a segment because they share flows; flow parameters are estimated from all pixels that have the same kind of flow. One significant advantage of this approach is that the estimated flow field can be discontinuous in space **exercises**.

35.3.3 Layered Motion: Estimating a Mixture of Affine Flows

A natural and widely applied flow model models image flow using multiple affine flows. Each pixel belongs to one of k segments. Each segment is associated with a set of affine flow parameters that explain its movement. This creates a flow model that admits discontinuities (like the simple mixture of flows) but where each

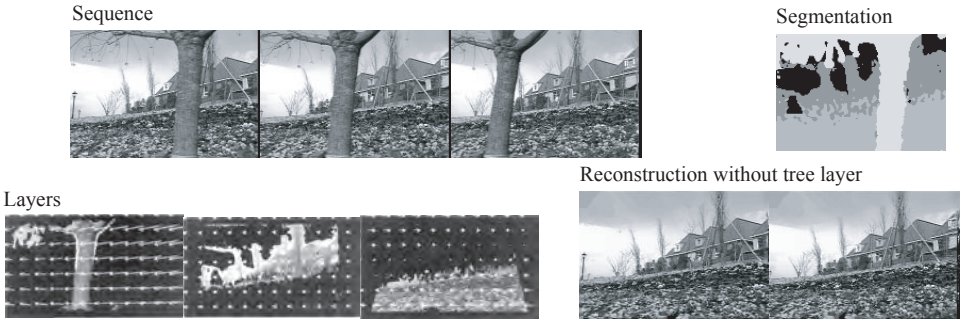

FIGURE 35.5: *Layered motion can be applied to longer sequences than just pairs of images (***top left***). You can think of the regions (***top right***) produced by layered motion as translucent layers with markings on them. Each layer moves independent of the others, and the layers are composed into the images (***bottom left***). When you do have a long sequence, you can recover the intensity of pixels that happen to be occluded in some (but not all) frames. In turn, this means you can build edited sequences where you compose only some of the layers (***bottom right***).* Image credit: *Fig. 6; part of Fig. 11; Fig. 12; Fig. 14 of "Representing Moving Images with Layers" by Wang and Adelson, 1994.*

particular flow can be quite complex. Just as in the previous section, if you know which flow model a pixel belongs to, you can estimate the parameters for that flow model by minimizing photometric error. Similarly, if you know the parameters of the flow models, you can tell which segment a pixel belongs to by checking which of the flow models minimizes the photometric error. Using the outline sketched in the previous section should seem reasonable to you.

The algorithm has one weakness. Each pixel is assigned to the segment whose flow yields the best photometric error. If two segments have about the same photometric error, this strategy could lose significant amounts of information. There is little justification for one segment's flow estimate getting all the constraint from that pixel, and the other segment's flow estimate getting none.

Improvements can be obtained with *soft assignment*, where each pixel can be assigned to more than one segment using a weight (the original strategy is then called *hard assignment*). To avoid overcounting, the weights must sum to one. The photometric error at a pixel yields a natural set of soft weights. Assume there are N different parametric models (equivalently, segments). Write θ_i for the parameters of the i'th segment, $\mathbf{f}(\mathbf{x}; i, \theta_i)$ for the flow predicted by the i'th model at $\mathbf{x}$ in the first frame and $E(\mathbf{x}; i, \theta_i) = (I_2(\mathbf{x} + \mathbf{f}(\mathbf{x}; i, \theta_i)) - I_1(\mathbf{x}))^2$ for the photometric error resulting from using the i'th segment to predict flow $\mathbf{x}$. Then a natural set of weights is

$$w_i(\mathbf{x}) = \frac{e^{\frac{-E(\mathbf{x};i,\theta_i)}{2\sigma^2}}}{\sum_u e^{\frac{-E(\mathbf{x};u,\theta_u)}{2\sigma^2}}}.$$

Here σ is a scale that you choose. Too small a value of σ and the soft assignment is largely a hard assignment because one weight is much larger than all the others.

Too large a value of σ and the soft assignment strategy oversmoothes the estimated flow field.

The result is known as *layered motion*. You segment the image into regions where everything in the region is subject to a particular affine flow. Segmenting an image sequence in this way means you can reassemble the segments in interesting ways (Figure 35.5).

35.4 YOU SHOULD

35.4.1 remember these definitions:

35.4.2 remember these facts:

35.4.3 remember these procedures:

35.4.4 be able to:

EXERCISES

QUICK CHECKS

35.1. Section 35.1.6 has: "you should expect reasonable results only for very small flows". Explain.

35.2. Procedure 35.1 remarks: " Notice that you will solve a linear system where for each $\mathbf{x}$, the matrix is the same. This means you could speed things up by forming the matrix, constructing a Cholesky or LU decomposition, and reusing that to solve the linear system at each point." Explain

35.3. Check that any movement of the camera where the focal point translates will result in a focus of expansion.

35.4. Section 34.2.2 has: "This flow is linear in image position. " Explain.

35.5. Section 34.2.2 has: "Notice the flow is zero when $(bx_2 + c) = 0$, which is also the horizon of the plane." Check this.

35.6. What is $\mathcal{M}$ for a constant flow? (notation in Section 34.2.5)

35.7. What is $\mathcal{M}$ for the flow of Section 34.2.2? (notation in Section 34.2.5)

35.8. Can you model the flow looking down on bumpy ground (Section 34.2.4) with an affine flow model?

What is $\mathcal{M}$ for the flow of Section **??**?(notation in Section 34.2.5)

35.9. Can you model the flow of Section 34.2.6) with an affine flow model?

35.10. Section 35.1 has: "Solving the resulting flow equation is straightforward, but you should expect reasonable results only for very small flows." Explain.

LONGER PROBLEMS

35.11. Assume that the image $\mathcal{I}$ in a $(2k + 1) \times (2k + 1)$ window around $\mathbf{x} = (i, j)$ is exactly modelled by

$$I_{i+u,j+v} = a_{11}u^2 + 2a_{12}uv + a_{22}v^2 + b_1u + b_2v + c.$$

Now translate this image by a small amount to get $\mathcal{J}$, so that $J_{i+t_x,j+t_y} = I_{i,j}$. For the moment, assume that t_x and t_y are integers.

(a) Show that the model

$$J_{i+u,j+v} = I_{i+u-t_x,j+v-t_y} = a_{11}(u-t_x)^2+2a_{12}(u-t_x)(v-t_y)+a_{22}(v-t_y)^2+b_1(u-t_x)+b_2(v-t_y)+c.$$

represents $\mathcal{J}$ around $\mathbf{x}$.

(b) How good is the representation of $\mathcal{J}$ if the flow is of the same size as the window?

(c) How good is the representation of $\mathcal{J}$ if the flow is considerably smaller than the image size, and $\mathbf{t}$ is not an integer?

(d) Assume this representation is very good. Write

$$J_{i+u,j+v} = a'_{11}u^2 + 2a'_{12}uv + a'_{22}v^2 + b'_1u + b'_2v + c.$$

Show that $a'_{11} = a_{11}$; $a'_{12} = a_{12}$; $a'_{22} = a_{22}$; $b'_1 = b_1 - 2a_{11}t_x - 2a_{12}t_y$; $b'_2 = b_2 - 2a_{12}t_x - 2a_{22}t_y$

(e) Use this information to explain why the text has:

> If $\mathcal{A}_1(\mathbf{x}_1)$, $\mathbf{b}_1(\mathbf{x}_1)$ and $c_1(\mathbf{x}_1)$ are an accurate local representation of $\mathcal{I}_1(\mathbf{x}_1)$, if $\mathcal{I}_2(\mathbf{x}) = \mathcal{I}_1(\mathbf{x}-\mathbf{t})$, and if $\mathbf{t}$ is sufficiently small, then
>
> $$\mathcal{A}_2 = \mathcal{A}_1,\ \mathbf{b}_2 = \mathbf{b}_1 - 2\mathcal{A}_1\mathbf{t}, \text{ and } c_2 = c_1 - \mathbf{b}_1^T\mathbf{t} + \mathbf{t}^T\mathcal{A}_1\mathbf{t}$$
>
> are an accurate representation of $\mathcal{I}_2(\mathbf{x}_1)$.

(f) Assume you have $\mathcal{A}_1(\mathbf{x})$, $\mathcal{A}_2(\mathbf{x})$, $\mathbf{b}_1(\mathbf{x})$, and $\mathbf{b}_2(\mathbf{x})$ representing a window at $\mathbf{x}$ in image 1 and image 2. Assume the flow $\mathbf{u}(\mathbf{x})$ at $\mathbf{x}$ is sufficiently small. Why does

$$(\mathcal{A}_1(\mathbf{x}) + \mathcal{A}_2(\mathbf{x}))\,\mathbf{u}(\mathbf{x}) = \mathbf{b}_2(\mathbf{x}) - \mathbf{b}_1(\mathbf{x})$$

yield an estimate of the flow?

PROGRAMMING EXERCISES