Dense Motion Estimation

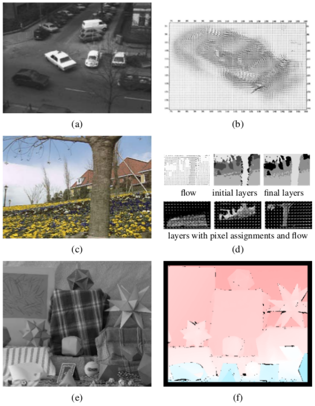
CSE 6367: Computer Vision

Instructor: William J. Beksi

Introduction

- Algorithms for aligning images and estimating motion in video sequences are among the most widely used in computer vision
- For example, frame-rate image alignment is widely used in camcorders and digital cameras to implement their image stabilization feature

Introduction



- Motion estimation: (a-b) regularization-based optical flow; (c-d) layered motion estimation; (e-f) sample image and ground truth flow from evaluation database

Aligning Two Images

- The simplest way to establish an alignment between two images or image patches is to shift one image relative to the other (**translational alignment**)
- Given a template image $I_0(\mathbf{x})$ sampled at discrete pixel locations $\{\mathbf{x}_i = (x_i, y_i)\}$, we wish to find where it is located in image $I_1(\mathbf{x})$

Aligning Two Images

- A least squares solution to this problem is to find the minimum of the sum of squared differences (SSD) function

$$E_{\mathrm{SSD}}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 = \sum_i e_i^2$$

  where $\mathbf{u} = (u, v)$ is the *displacement* and $e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)$ is called the *residual error*

- The assumption that corresponding pixel values remain the same in the two images is often called the **brightness constancy constraint**

Aligning Two Images

- In general, the displacement $\mathbf{u}$ can be fractional, so a suitable interpolation function must be applied to image $I_1(\mathbf{x})$
- In practice, a bilinear interpolant is often used but bicubic interpolation can yield slightly better results
- Color images can be processed by summing differences across all three color channels, or by first transforming the images into a different color space, or by only using the luminance

Robust Error Metrics

- We can make residual error metric more robust to outliers by replacing the squared error terms with a robust function $\rho(e_i)$ to obtain

$$E_{\mathrm{SRD}}(\mathbf{u}) = \sum_i \rho(\mathbf{I}_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)) = \sum_i \rho(e_i)$$

- The robust norm $\rho(e)$ is a function that grows less quickly than the quadratic penalty associated with least squares

Robust Error Metrics

- The sum of absolute differences (SAD), sometimes used for video coding because of its speed, uses an L_1 norm

$$E_{\text{SAD}}(\mathbf{u}) = \sum_i |I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)| = \sum_i |e_i|$$

- However, since this function is not differentiable at the origin it is not well suited to gradient descent approaches

Robust Error Metrics

- We'd like to use a smoothly varying function that is quadratic for small values but grows more slowly away from the origin
- One often used way to do this is the Geman-McClure function

$$\rho_{\text{GM}}(x) = \frac{x^2}{1 + x^2/a^2}$$

where a is a constant that can be thought of as an outlier threshold

Spatially Varying Weights

- The previously mentioned error metrics ignore the fact that for a given alignment, some of the pixels being compared may lie outside the original image boundaries
- Furthermore, we many want to partially or completely downweight the contributions of certain pixels
- For applications such as background stabilization, we may want to downweight the middle part of the image which often contains independently moving objects being tracked by the camera

Spatially Varying Weights

- All of these tasks can be accomplished by associating a spatially varying per-pixel weight value with each of the two images being matched
- The error metric then becomes the weighted (or windowed) SSD function

$$E_{\text{WSSD}}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i) w_1(\mathbf{x}_i + \mathbf{u}) [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2$$

where the weighting functions w_0 and w_1 are zero outside the image boundaries

Spatially Varying Weights

- If a large range of potential motions is allowed, the E_{WSSD} metric can have a bias towards smaller overlap solutions
- To counteract this bias, the windowed SSD score can be divided by the overlap area

$$A = \sum_i w_0(\mathbf{x}_i) w_1(\mathbf{x}_i + \mathbf{u})$$

to compute a per-pixel (or mean) squared pixel error E_{WSSD}/A where the square root of this quantity is known as the root mean square intensity error

$$RMS = \sqrt{E_{\text{WSSD}}/A}$$

Bias and Gain (Exposure Differences)

- Often, two images being aligned were not taken with the same exposure
- A simple model of linear (affine) intensity variation between the two images is the bias and gain model

$$I_1(\mathbf{x} + \mathbf{u}) = (1 + \alpha) I_0(\mathbf{x}) + \beta$$

  where β is the bias and α is the gain

- The least squares formulation then becomes

$$\begin{aligned} E_{\mathrm{BG}}(\mathbf{u}) &= \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - (1 + \alpha) I_0(\mathbf{x}_i) - \beta]^2 \\ &= \sum_i [\alpha I_0(\mathbf{x}_i) + \beta - e_i]^2 \end{aligned}$$

Bias and Gain (Exposure Differences)

- Rather than taking a simple squared difference between corresponding patches, it becomes necessary to perform a linear regression which is somewhat more costly
- Note that for color images, it may be necessary to estimate a different bias and gain for each color channel to compensate for the automatic color correction performed by some digital cameras

Correlation

- An alternative to taking intensity differences is to perform correlation, i.e. to maximize the product, or cross-correlation (CC), of the two aligned images

$$E_{\mathrm{CC}}(\mathbf{u}) = \sum_i I_0(\mathbf{x}_i) I_1(\mathbf{x}_i + \mathbf{u})$$

- This may appear to make bias and gain modeling unnecessary since the images will prefer to line up regardless of their relative scales and offsets

- However, if a very bright patch exists in $I_1(\mathbf{x})$ the maximum product may actually lie in that area

Correlation

- For this reason, normalized cross-correlation (NCC) is more commonly used

$$E_{\text{NCC}}(\mathbf{u}) = \frac{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0][I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]}{\sqrt{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0]^2 [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]^2}}$$

where $\bar{I}_0 = \frac{1}{N}\sum_i I_0(\mathbf{x}_i)$ and $\bar{I}_1 = \frac{1}{N}\sum_i I_1(\mathbf{x}_i + \mathbf{u})$ are the mean images of the corresponding patches and N is the number of pixels in the patch

Correlation

- The NCC score is always guaranteed to be in the range $[-1, 1]$ which makes it easier to handle in some higher-level applications such as deciding which patches truly match
- NCC works well when matching images taken with different exposures, e.g. when creating high dynamic range images
- Note that the NCC score is undefined if either of the two patches has zero variance (and in fact its performance degrades for noisy low-contrast regions)

Hierarchical Motion Estimation

- Now that we have a well-defined alignment cost function to optimize, how can we find its minimum?
- The simplest solution is to do a full search over some range of shifts using either integer or sub-pixel steps
- This is often the approach used for block matching in motion compensated video compression where a range of possible motions (e.g. ± 16 pixels) is explored

Hierarchical Motion Estimation

- To accelerate this search process, hierarchical motion estimation is often used: an image pyramid is constructed and a search over a smaller number of discrete pixels is first performed at coarser levels
- The motion estimate from one level of the pyramid is then used to initialize a smaller local search at the next finer level

Hierarchical Motion Estimation

- More formally, let

$$I_k^{(l)}(\mathbf{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\mathbf{x}_j)$$

  be the decimated image at level l obtained by subsampling a smoothed version of the image at level $l-1$

- At the coarsest level, we search for the best displacement $\mathbf{u}^{(l)}$ that minimizes the difference between images $I_0^{(l)}$ and $I_1^{(l)}$

Hierarchical Motion Estimation

- This is usually done using a full search over some range of displacements $\mathbf{u}^{(l)} \in 2^{-l}[-S, S]^2$, where S is the desired search range at the finest (original) resolution level
- Once a suitable motion vector has been estimated, it is used to predict a likely displacement

$$\hat{\mathbf{u}}^{(l-1)} \leftarrow 2\mathbf{u}^{(l)}$$

  for the next finer level
- The search over displacements is then repeated at the finer level over a much narrower range of displacements (e.g. $\hat{\mathbf{u}}^{(l-1)} \pm 1$)

Fourier-based Alignment

- When the search range corresponds to a significant fraction of the larger image (e.g. image stitching), the hierarchical approach may not work that well
- This is because it is often not possible to coarsen the representation too much before significant features are blurred away
- In this case, a Fourier-based approach may be preferable

Fourier-based Alignment

- Fourier-based alignment relies on the fact that the Fourier transform of a shifted signal has the same magnitude as the original signal, but a linearly varying phase, i.e.

$$\mathcal{F}\{I_1(\mathbf{x}+\mathbf{u}\} = \mathcal{F}\{I_1(\mathbf{x})\}e^{-ju\cdot\boldsymbol{\omega}} = \mathcal{I}_1(\boldsymbol{\omega})e^{-ju\cdot\boldsymbol{\omega}}$$

  where $\boldsymbol{\omega}$ is the vector-valued angular frequency of the Fourier transform and we use $\mathcal{I}_1(\boldsymbol{\omega}) = \mathcal{F}\{I_1(\mathbf{x})\}$ to denote the Fourier transform of a signal

Fourier-based Alignment

- Another useful property of Fourier transforms is the convolution in the spatial domain corresponds to multiplication in the Fourier domain
- Thus, the Fourier transform of the cross-correlation E_{CC} can be written as

$$\mathcal{F}\{E_{CC}(\mathbf{u})\} = \mathcal{F}\left\{\sum_i I_0(\mathbf{x}_i) I_1(\mathbf{x}_i + \mathbf{u})\right\} = \mathcal{F} I_0(\mathbf{u}) \bar{*} I_1(\mathbf{u}) = \mathcal{I}_0(\omega)\mathcal{I}_1^*(\omega)$$

where $f(\mathbf{u})\bar{*}g(\mathbf{u}) = \sum_i f(\mathbf{x}_i)g(\mathbf{x}_i + \mathbf{u})$ is the correlation function, i.e. the convolution of one signal with the reverse of the other, and $\mathcal{I}_1^*(\omega)$ is the complex conjugate of $\mathcal{I}_1(\omega)$

Fourier-based Alignment

- Thus, to efficiently evaluate E_{CC} over the range of all possible values of $\mathbf{u}$, we take $I_0(\mathbf{x})$ and $I_1(\mathbf{x})$, multiply them together, and take the inverse transform of the result
- The Fast Fourier Transform (FFT) algorithm can compute the transform of an $N \times M$ image in $O(NM \log NM)$ operations
- This can be significantly faster than the $O(N^2M^2)$ operations required to do a full search when the full range of overlaps is considered

Fourier-based Alignment

- While Fourier-based convolution if often used to accelerate the computation of image correlations, it can also be used to accelerate the SSD function

- The Fourier transform of the SSD formula can be written as

$$\begin{aligned}\mathcal{F}\{E_{\mathrm{SSD}}(\mathbf{u})\} &= \mathcal{F}\left\{\sum_i [I_1(\mathbf{x}_i+\mathbf{u}) - I_0(\mathbf{x}_i)]^2\right\} \\ &= \delta(\boldsymbol{\omega})\sum_i [I_0^2(\mathbf{x}_i) + I_1^2(\mathbf{x}_i] - 2\mathcal{I}_0(\boldsymbol{\omega})\mathcal{I}_1^*(\boldsymbol{\omega})\end{aligned}$$

- Thus, the SSD can be computed by taking twice the correlation function and subtracting it from the sum of the energies in the two images

Windowed Correlation

- Unfortunately, the Fourier convolution theorem only applies when the summation over $\mathbf{x}_i$ is performed over *all* pixels in both images using a circular shift of the image when accessing pixels outside the original boundaries
- While this is acceptable for small shifts and comparably sized images, it makes no sense when the images overlap by a small amount or one image is a small subset of the other

Windowed Correlation

- In that case, the CC function should be replaced with a windowed (weighted) CC function

$$E_{\text{WCC}}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i) I_0(\mathbf{x}_i) + w_1(\mathbf{x}_i + \mathbf{u}) I_1(\mathbf{x}_i + \mathbf{u})$$
$$= [w_0(\mathbf{x}) I_0(\mathbf{x})] \bar{*} [w_1(\mathbf{x}) I_1(\mathbf{x})]$$

where the weighting functions w_0 and w_1 are zero outside the valid ranges of images and both images are padded so that circular shifts return 0 values outside the original image boundaries

Rotations and Scale

- While Fourier-based alignment is mostly used to estimate translational shifts between images, it can, under certain limited conditions, also be used to estimate in-plane rotations and scales
- Consider two images that are related purely by rotation, i.e.

$$I_1(\hat{R}\mathbf{x}) = I_0(\mathbf{x})$$

Rotations and Scale

- If we resample the images into polar coordinates

  $$\tilde{I}_0(r,\theta) = I_0(r\cos\theta, r\sin\theta) \quad \text{and} \quad \tilde{I}_1(r,\theta) = I_1(r\cos\theta, r\sin\theta)$$

  we obtain

  $$\tilde{I}_1(r,\theta+\hat{\theta}) = I_0(r,\theta)$$

- The desired rotation can then be estimated using an FFT shift-based technique

Rotations and Scale

- If the two images are also related by a scale

  $$I_1(e^{\hat{s}}\hat{R}\mathbf{x}) = I_0(\mathbf{x})$$

  we can resample into log-polar coordinates

  $$\tilde{I}_0(s,\theta) = I_0(e^s\cos\theta, e^s\sin\theta) \quad \text{and} \quad \tilde{I}_1(s,\theta) = I_1(e^s\cos\theta, e^s\sin\theta)$$

  to obtain

  $$\tilde{I}_1(s+\hat{s}, \theta+\hat{\theta}) = I_0(s,\theta)$$

- In this case, care must be taken to choose a suitable range of s values that resonably samples the original image

Incremental Refinement

- The techniques mentioned so far can estimate alignment to the nearest pixel (or potentially fractional pixel if smaller search steps are used)
- In general, image stabilitizaton and stitching applications require much higher accuracies to obtain acceptable results

Incremental Refinement

- A commonly used approach, first introduced by Lucas and Kanade (1981), is to perform gradient descent on the SSD energy function using a Taylor series expansion of the image function

$$\begin{aligned} E_{\text{LK-SSD}}(\mathbf{u} + \Delta\mathbf{u}) &= \sum_i [I_1(\mathbf{x}_i + \mathbf{u} + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2 \\ &\approx \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) + J_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} - I_0(\mathbf{x}_i)]^2 \\ &= \sum_i [J_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i]^2 \qquad (1) \end{aligned}$$

Incremental Refinement

where

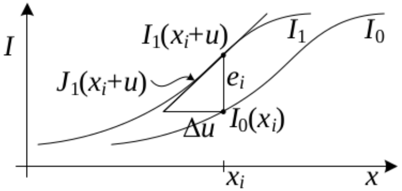
$$J_1(\mathbf{x}_i + \mathbf{u}) = \nabla I_1(\mathbf{x}_i + \mathbf{u}) = (\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y})(\mathbf{x}_i + \mathbf{u})$$

is the image gradient or Jacbobian at $(\mathbf{x}_i + \mathbf{u})$ and

$$e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)$$

is the current intensity error

Incremental Refinement



- Taylor series approximation of a function and the incremental computation of the optical flow correction amount: $J_1(\mathbf{x}_i + \mathbf{u})$ is the image gradient at $(\mathbf{x}_i + \mathbf{u})$ and e_i is the current intensity difference

Incremental Refinement

- The gradient at a particular sub-pixel location $(\mathbf{x}_i + \mathbf{u})$ can be computed using a variety of techniques
- The simplest method is to simply take the horizontal and vertical differences between pixels $\mathbf{x}$ and $\mathbf{x} + (1, 0)$ or $\mathbf{x} + (0, 1)$

Incremental Refinement

- The linearized form of the incremental update to the SSD error is often called the *optical flow* or *brightness constancy constraint* equation

$$I_x u + I_y v + I_t = 0$$

  where the subscripts in I_x and I_y denote the spatial derivatives, and I_t is called the temporal derivative

- When squarred and summed or integrated over a region, it can be used to compute optic flow

Incremental Refinement

- The least squares problem, equation (1), can be minimized by solving the associated normal equations

$$A\Delta\mathbf{u} = \mathbf{b}$$

where

$$A = \sum_i J_1^T(\mathbf{x}_i + \mathbf{u})J_1(\mathbf{x}_i + \mathbf{u})$$

and

$$b = -\sum_i e_i J_1^T(\mathbf{x}_i + \mathbf{u})$$

are called the (Gauss-Newton approximation of the) Hessian and gradient-weighted residual vector, respectively

Incremental Refinement

- These matrices are often written as

$$A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \quad \text{and} \quad b = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

- The gradients required for $J_1(\mathbf{x}_i + \mathbf{u})$ can be evaluated at the same time as the image warps required to estimate $I_1(\mathbf{x}_i + \mathbf{u})$, and in fact are often computed as a side product of image interpolation

Incremental Refinement

- If efficiency is a concern, these gradients can be replaced by the gradients in the *template* image

$$J_1(\mathbf{x}_i + \mathbf{u}) \approx J_0(\mathbf{x}_i)$$

  since near the correct alignment, the template and displaced target images should look similar

- This has the advantage of allowing the precomputation of the Hessian and Jacobian images, which can result in significant computational savings

Incremental Refinement

- The effectiveness of the incremental update rule relies on the quality of the Taylor series approximation
- When far away from the true displacement (e.g. 1-2 pixels), several iterations may be needed
- However, it is possible to estimate a value for J_1 using a least squares fit to a series of larger displacements in order to increase the range of convergence or to “learn” a special-purpose recognizer for a given patch

Incremental Refinement

- A commonly used stopping criterion for incremental updating is to monitor the magnitude of a displacement correction $||\mathbf{u}||$ and to stop when it drops below a certain threshold (e.g. 1/10 of a pixel)
- For larger motions, it is usual to combine the incremental update rule with a hierarchical coarse-to-fine search strategy

Lucas-Kanade Algorithm

- Many image alignment tasks, for example image stitching with handheld cameras, require the use of more sophisticated motion models
- Since these models, e.g. affine deformations, typically have more parameters than pure translation, a full search over the possible range of values impractical
- Instead, the incremental Lucas-Kanade algorithm can be generalized to **parametric motion** models and used in conjunction with a hierarchical search algorithm

Lucas-Kanade Algorithm

- For parametric motion, instead of using a single constant translation vector $\mathbf{u}$ we use a spatially varying motion field or correspondence map, $\mathbf{x}'(\mathbf{x}; \mathbf{p})$
- It is parameterized by a low-dimensional vector $\mathbf{p}$, where $\mathbf{x}'$ can be any of the 2D planar transformation motion models (e.g. translation, rotation, etc.)

Lucas-Kanade Algorithm

- The parametric incremental motion update rule now becomes

$$\begin{aligned} E_{\text{LK-PM}}(\mathbf{p} + \Delta\mathbf{p}) &= \sum_i [I_1(\mathbf{x}'(\mathbf{x}_i; \mathbf{p} + \Delta\mathbf{p})) - I_0(\mathbf{x}_i)]^2 \\ &\approx \sum_i [I_1(\mathbf{x}'_i) + J_1(\mathbf{x}'_i)\Delta\mathbf{p} - I_0(\mathbf{x}_i)]^2 \\ &= \sum_i [J_1(\mathbf{x}'_i)\Delta\mathbf{p} + e_i]^2 \end{aligned}$$

where the Jacobian is now

$$J_1(\mathbf{x}'_i) = \frac{\partial I_1}{\partial \mathbf{p}} = \nabla I_1(\mathbf{x}'_i)\frac{\mathbf{x}'}{\partial \mathbf{p}}(\mathbf{x}_i)$$

i.e. the product of the image gradient ∇I_1 with the Jacobian of the correspondence field, $J_{x'} = \partial\mathbf{x}'/\partial\mathbf{p}$

Lucas-Kanade Algorithm

- The (Gauss-Newton) and gradient-weighted residual vector for parametric motion becomes

$$A = \sum_i J_{\mathbf{x}'}^T(\mathbf{x}_i)[\nabla I_1^T(\mathbf{x}_i')\nabla I_1(\mathbf{x}_i')]J_{\mathbf{x}'}(\mathbf{x}_i)$$

and

$$b = -\sum_i J_{\mathbf{x}'}^T(\mathbf{x}_i)[e_i \nabla I_1^T(\mathbf{x}_i')]$$

Patch-based Approximation

- The computation of the Hessian and residual vectors for parametric motion can be significantly more expensive than for the translational case
- For parametric motion with n parameters and N pixels, the accumulation of A an b takes $O(n^2N)$ operations

Patch-based Approximation

- One way to reduce this by a significant amount is to divide the image up into smaller sub-blocks (patches) P_j and to only accumulate the simpler 2×2 quantities inside the square brackets at the pixel level

$$A = \sum_{i \in P_j} \nabla I_1^T(\mathbf{x}_i') \nabla I_1(\mathbf{x}_i')$$

$$b = \sum_{i \in P_j} e_i \nabla I_1^T(\mathbf{x}_i')$$

Patch-based Approximation

- The full Hessian and residual can then be approximated as

$$\begin{aligned} A &\approx \sum_j J_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j)[\sum_{i \in P_j} \nabla I_1^T(\mathbf{x}_i')\nabla I_1(\mathbf{x}_i')]J_{\mathbf{x}'}(\hat{\mathbf{x}}_j) \\ &= \sum_j J_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) A_j J_{\mathbf{x}'}(\hat{\mathbf{x}}_j) \end{aligned}$$

and

$$\begin{aligned} b &\approx -\sum_j J_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j)[\sum_{i \in P_j} e_i \nabla I_1^T(\mathbf{x}_i')] \\ &= -\sum_j J_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j)\mathbf{b}_j \end{aligned}$$

where $\hat{\mathbf{x}}_j$ is the *center* of each patch P_j

Compositional Approach

- For a complex parametric motion such as a homography, the computation of the motion Jacobian becomes complicated and may involve a per-pixel division
- This can be simplified by first warping the target image I_1 according to the current motion estimate $\mathbf{x}'(\mathbf{x};\mathbf{p})$

$$\tilde{I}_1(\mathbf{x}) = I_1(\mathbf{x}'(\mathbf{x};\mathbf{p}))$$

and then comparing this *warped* image against the template $I_0(\mathbf{x})$

$$\begin{aligned} E_{\text{LK-SS}}(\Delta\mathbf{p}) &= \sum_i [\tilde{I}_1(\tilde{\mathbf{x}}(\mathbf{x}_i;\Delta\mathbf{p})) - I_0(\mathbf{x}_i)]^2 \\ &\approx \sum_i [\tilde{J}_1(\mathbf{x}_i)\Delta\mathbf{p} + e_i]^2 \\ &= \sum_i [\nabla\tilde{I}_1(\mathbf{x}_i) J_{\tilde{\mathbf{x}}}(\mathbf{x}_i)\Delta\mathbf{p} + e_i]^2 \end{aligned}$$

Compositional Approach

- Note that since the two images are assumed to be fairly similar, only an *incremental* parametric motion model is required
- In other words, the incremental motion can be evaluated around $\mathbf{p} = 0$ which can lead to considerable simplifications

Compositional Approach

- For example, the Jacobian of the planar projective transform now becomes

$$J_{\tilde{\mathbf{x}}} = \frac{\partial \tilde{\mathbf{x}}}{\partial p}\bigg|_{\mathbf{p}=0} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}$$

- Once the incremental motion $\tilde{\mathbf{x}}$ has been computed, it can be *prepended* to the previously estimated motion

Application: Video Stabilization

- Video stabilization is one of the most widely used applications of parametric motion estimation
- Algorithms for stabilization run inside both hardware devices and software packages for improving the quality of shaky videos

Application: Video Stabilization

- In full-frame video stabilization there are three major stages: motion estimation, motion smoothing, and image warping
- Motion estimation algorithms often use a similarity transform to handle camera translations, rotations, and zooming
- The tricky part is getting these algorithms to lock onto the background motion, which is a result of camera movement, without getting distracted by independent moving foreground objects

Application: Video Stabilization

- Motion smoothing algorithms recover the low-frequency (slowly varying) part of the motion and then estimate the high-frequency shake component that needs to be removed
- Finally, image warping algorithms apply the high-frequency correction to render the original frames as if the camera had undergone only the smooth motion

Application: Video Stabilization

- The resulting stabilization algorithms can greatly improve the appearance of shaky videos, but they often still contain visual artifacts
- For example, image warping can result in missing borders around the image which must be cropped and filled using information from other frames
- Furthermore, video frames captured during fast motion are often blurry and need to be improved using deblurring techniques or stealing sharper pixels from other frames with less motion or better focus

Application: Video Stabilization

- In situations where the camera is translating a lot in 3D, e.g. when the videographer is walking, an even better approach is to compute a full structure from motion reconstruction of the camera motion and 3D scene
- A smooth 3D camera path can then be computed and the original video re-rendered using view interpolation with the interpolated 3D point cloud serving as the proxy geometry while preserving salient features

Learned Motion Models

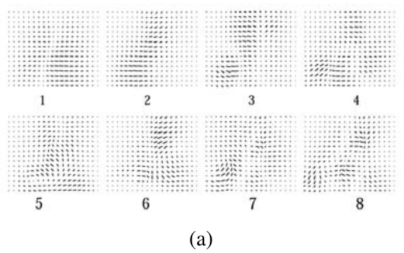
- An alternative to parameterizing the motion field with a geometric deformation such as an affine transform is to learn a set of basis functions tailored to a particular application
- To do this, we first compute a set of dense motion fields from a set of training videos

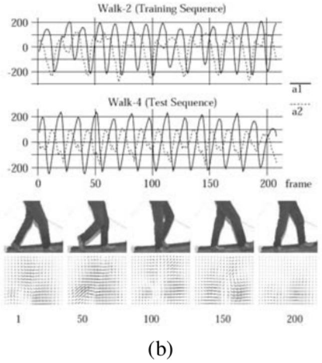
Learned Motion Models

- Next, SVD is applied to the stack of motion fields $\mathbf{u}_t(\mathbf{x})$ to compute the first few singular vectors $\mathbf{v}_k(\mathbf{x})$
- Finally, for a new test sequence, a novel flow field is computed using a coarse-to-fine algorithm that estimates the unknown coefficient a_k in the parameterized flow field

$$\mathbf{u}(\mathbf{x}) = \sum_k a_k \mathbf{v}_k(\mathbf{x})$$

Learned Motion Models





- Learned parameterized motion fields for a walking sequence: (a) learned basis flow fields; (b) plots of motion coefficients over time and corresponding estimated motion fields

Spline-based Motion Estimator

- While parametric motion models are useful in a wide variety of applications (such as video stabilization and mapping onto planar surfaces), most image motion is too complicated to be captured by such low-dimensional models
- Traditionally, optical flow algorithms compute an independent motion estimate for each pixel, i.e. the number of flow vectors computed is equal to the number of input pixels

Spline-based Motion Estimator

- The general optical flow analog to E_{SSD} can be written as

$$E_{\text{SSD-OF}}(\{\mathbf{u}_i\}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}_i) - I_0(\mathbf{x}_i)]^2$$

- Note that the number of variables $\{\mathbf{u}_i\}$ is twice the number of measurements, so the problem is underconstrained

Spline-based Motion Estimator

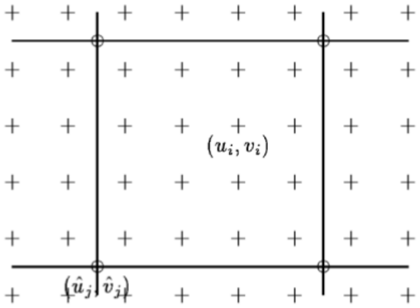
- Our approach to this problem is to represent the motion field as a 2D **spline** controlled by a smaller number of **control vertices** $\{\mathbf{u}_i\}$

$$\mathbf{u}_i = \sum_j \hat{\mathbf{u}}_j B_j(\mathbf{x}_i) = \sum_j \hat{\mathbf{u}}_j w_{i,j}$$

  where the $B_j(\mathbf{x}_i)$ are called the **basis functions** and are only non-zero over a small *finite support* interval

- We call the $w_{i,j} = B_j(\mathbf{x}_i)$ **weights** to emphasize that the $\{\mathbf{u}_i\}$ are known linear combinations of the $\{\hat{\mathbf{u}}_j\}$

Spline-based Motion Estimator



- Spline motion field: the displacement vectors $\mathbf{u}_i = (u_i, v_i)$ are shown as pluses $(+)$ and are controlled by the smaller number of control vertices $\hat{\mathbf{u}}_j = (\hat{u}_i, \hat{v}_j)$, which are shown as circles

Spline-based Motion Estimator

- Substituting the formula for the individual per-pixel flow vectors $\mathbf{u}_i$ into the SSD error metric yields a parametric motion formula similar to the Lucas-Kanade parametric incremental update rule
- The biggest difference is that the Jacobian $J_1(\mathbf{x}'_i)$ now consists of the sparse entries in the weight matrix $W = [w_{i,j}]$

Spline-based Motion Estimator

- In many cases, the small number of spline vertices results in a motion estimation problem that is well conditioned
- However, if large textureless regions persist across several spline patches then it may be necessary to add a *regularization* term to make the problem well posed

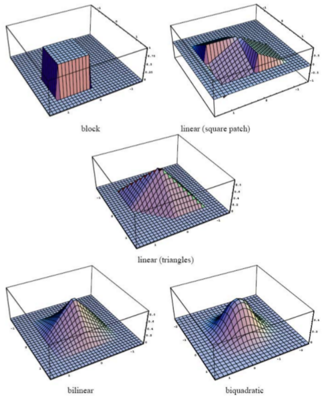
Spline-based Motion Estimator

- The simplest way to do this is to directly add squared difference penalties between adjacent vertices in the spline control mesh $\{\hat{\mathbf{u}}_j\}$
- If a multi-resolution (coarse-to-fine) strategy is being used, it is important to rescale these smoothness terms while going from level to level

Spline-based Motion Estimator

- The linear system corresponding to the spline-based motion estimator is sparse and regular
- Since it is usually of moderate size, it can often be solved using direct techniques such as Cholesky decomposition
- Alternatively, if the problem becomes too large and subject to excessive fill-in, iterative techniques such as hierarchically preconditioned conjugate gradient can be used instead

Spline-based Motion Estimator

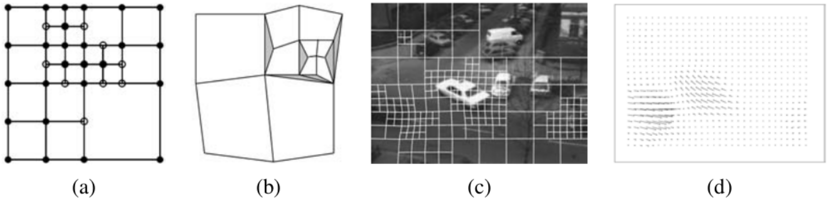


- Sample spline basis functions: the block (constant) interpolator/basis corresponds to block-based motion estimation

Spline-based Motion Estimator

- Due to its robustness, spline-based motion estimation has been used for a number of applications, including visual effects and medical image registration
- One disadvantage of the basic technique is that the model does a poor job near motion discontinuities unless an excessive number of nodes is used
- To remedy this situation, a quadtree representation embedded in the spline control grid can be used

Spline-based Motion Estimator

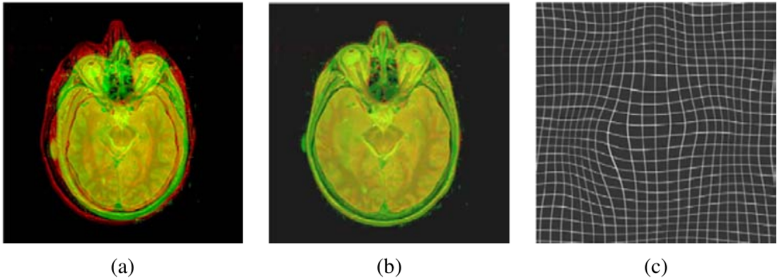


- Quadtree spline-based motion estimation: (a) quadtree spline representation, (b) which can lead to cracks, unless the white nodes are constrained to depend on their parents; (c) deformed quadtree spline mesh overlaid on grayscale image; (d) flow field visualized as a needle diagram

Application: Medical Image Registration

- Since they excel at representing smooth *elastic* deformation fields, spline-based motion models have found widespread use in medical image registration
- Registration techniques can be used both to track an individual patient's development or progress over time
- They can also be used to match different patient images together to find commonalities and detect variations or pathologies

Application: Medical Image Registration



- Elastic brain registration: (a) original brain atlas and patient MRI images overlaid in red-green; (b) after elastic registration with eight user specified landmarks (not shown); (c) a cubic B-spline deformation field, shown as a deformed grid

Estimating Pixel Motion

- The most general (and challenging) version of motion estimation is to compute an independent estimate of motion at *each* pixel
- This is generally known as **optical** (or **optic**) **flow**

Estimating Pixel Motion

- It generally involves minimizing the brightness or color difference between corresponding pixels summed over the image

$$E_{\text{SSD-OF}}(\{\mathbf{u}_i\}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}_i) - I_0(\mathbf{x}_i)]^2 \tag{2}$$

- The two classic approaches to this problem are (1) perform the summation *locally* over overlapping regions (the patch-based or window-based approach); (2) add smoothness terms on the $\{\mathbf{u}_i\}$ field using regularization or Markov random fields and to search for a global minimum

Patch-based Approach

- The patch-based approach usually involves using a Taylor series expansion of the displaced image function in order to obtain sub-pixel estimates
- A series of local discrete search steps can be interleaved with Lucas-Kanade incremental refinement steps in a coarse-to-fine pyramid scheme

Regularization-based Approach

- Instead of solving for each motion (or motion update) independently, Horn and Schunck (1981) develop a regularization-based framework where equation (2) is simultaneously minimized over all flow vectors $\{\mathbf{u}_i\}$
- In order to constrain the problem, smoothness constraints, i.e. squared penalties on flow derivatives, are added to the basic per-pixel error metric

Regularization-based Approach

- This technique was originally developed for small motions in a variational (continuous function) framework
- Therefore, the linearized brightness constancy constraint is more commonly written as an analytic integral

$$E_{\mathrm{HS}} = \int (I_x u + I_y v + I_t)^2 dx\, dy$$

  where $(I_x, I_y) = \nabla I_1 = J_1$ and $I_t = e_i$ is the temporal derivative, i.e. the brightness change between images
- The Horn and Schunck model can also be viewed as the limiting case of spline-based motion estimation as the splines become 1×1 pixel patches

Optical Flow via Convolutional Neural Networks

- Recently, there has been an interest in using convolutional neural networks and deep networks to develop supervised and unsupervised techniques for estimating optical flow
- Optical flow estimation needs both precise per-pixel localization and the ability to find correspondences between two input images which can be cast as a learning problem
- This is an active area of research: FlowNet, FlowNet 2.0, etc.

Multi-frame Motion Estimation

- We've looked at motion estimation as a two-frame problem where the goal is to compute a motion field that aligns pixels from one image with those in another
- In practice, motion estimation is usually applied to video where a whole sequence of frames is available to perform this task

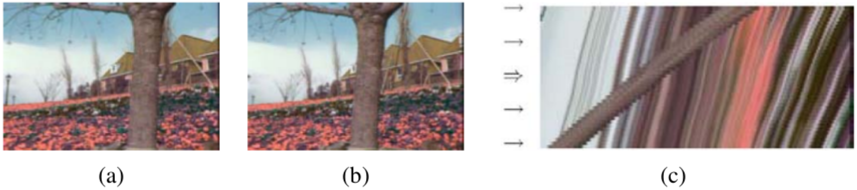
Multi-frame Motion Estimation

- A classic approach to multi-frame motion is to *filter* the spatio-temporal volume using oriented or steerable filters
- This is done in a manner analogous to oriented edge detection
- Spatio-temporal filtering uses a 3D volume around each pixel to determine the best orientation in space-time which corresponds directly to a pixel's velocity

Multi-frame Motion Estimation

- Unfortunately, in order to obtain reasonably accurate velocity estimates everywhere in an image, spatio-temporal filters have moderately large extents, which severely degrades the quality of their estimates near motion discontinuities
- An alternative to full spatial-temporal filtering is to estimate more local spatio-temporal derivatives and use them inside a global optimization framework to fill in textureless regions

Multi-frame Motion Estimation

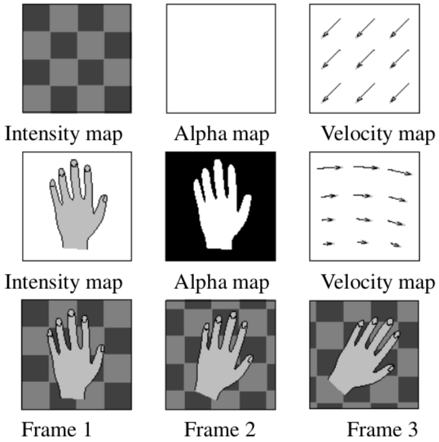


- Slice through a spatio-temporal volume: (a-b) two frames from the flower garden sequence; (c) a horizontal slice through the complete spatio-temporal volume, with the arrows indicating locations of potential key frames where flow is estimated

Layered Motion Representations

- Visual motion if often caused by the movement of a small number of objects at different depths in the scene
- In such situations, the pixel motions can be described more succinctly (and estimated more reliably) if pixels are grouped into appropriate objects or **layers**

Layered Motion Representations



- Layered motion estimation framework

Layered Motion Representations

- The motion in this sequence is caused by the translational motion of the checkered background and the rotation of the foreground hand
- The complete motion sequence can be reconstructed from the appearance of the foreground and background elements, which can be represented as alpha-matted images and the parametric motion corresponding to each layer
- Displacing and compositing in back to front order recreates the original video sequence

Layered Motion Representations

- Layered motion representations not only lead to compact representations, but they also exploit the information available in multiple video frames along with accurately modeling the appearance of pixels near motion discontinuities
- This makes them particularly suited as a representation for image-based rendering as well as object-level video editing

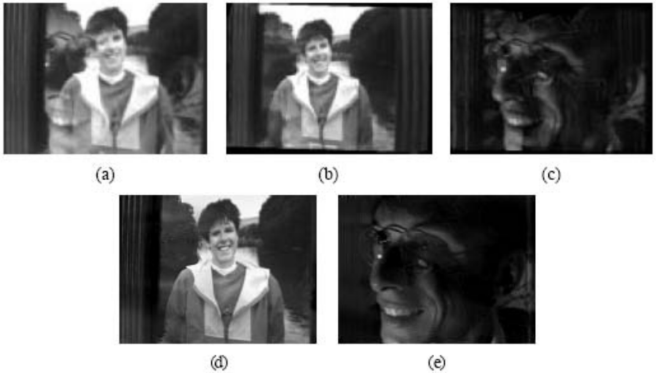
Transparent Layers and Reflections

- A special case of layered motion that occurs quite often is transparent motion caused by reflections seen in windows and picture frames
- Due to the way that light is both reflected from and transmitted through a glass surface, a model to separate transparent layers must be an additive one (i.e. each moving layer contributes some intensity to the final image)

Transparent Layers and Reflections

- If the motions of the individual layers are known, the recovery of the individual layers is a simple constrained least squares problem, where the individual layer images are constrained to be positive
- However, this problem can suffer from extended low-frequency ambiguities, especially if either of the layers lacks dark (black) pixels or the motion is uni-directional

Transparent Layers and Reflections



- Light reflecting off the transparent glass of a picture frame: (a) first image from the input sequence; (b) dominant motion layer min-composite; (c) secondary motion residual layer max-composite; (d-e) final estimated picture and reflection layers

Summary

- To estimate the motion between two or more images, a suitable error metric must first be chosen to compare the images
- Once this has been established, a suitable search technique must be devised (full search, hierarchical coarse-to-fine, Fourier transforms, etc.)
- To get sub-pixel precision in the alignment, incremental methods based on a Taylor series expansion of the image function can be used
- Recent research makes use of deep networks for estimating optical flow