Back to Lucas-Kanade

Step-by-Step Derivation

First consider the expansion for a single variable $p$

$$E_p = E(y(x, p + △p)) = E(y(x, p)) + \frac{∂E}{∂y} \Delta y + \frac{∂E}{∂p} \Delta p$$

Step-by-Step Derivation

Now let’s rewrite the expression as a matrix equation. For each term, we can rewrite:

$$\frac{∂E}{∂y} \Delta y = \begin{bmatrix} ∂E/∂y_1 & ∂E/∂y_2 & ... & ∂E/∂y_n \end{bmatrix} \Delta y$$

So that we have:

$$E_p = \begin{bmatrix} \frac{∂E}{∂y} & \frac{∂E}{∂p} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta p \end{bmatrix}$$

Step-by-Step Derivation

Further collecting the $dw/dp$ terms into a matrix, we can write:

$$E_p = \begin{bmatrix} \frac{∂E}{∂y} & \frac{∂E}{∂p} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta p \end{bmatrix}$$

Note that each variable parameter $p_i$ contributes a term of the form

$$\frac{∂E}{∂y_i} \Delta y_i$$

Step-by-Step Derivation

The key to the derivation is Taylor series approximation:

$$E(W(x+y), P + △P) \approx E(W(x+y), P) + \frac{∂E}{∂W} \Delta W$$

We will derive this step-by-step. First, we need two background formula:

$$\frac{∂E}{∂W} = \begin{bmatrix} \frac{∂E}{∂y_1} & \frac{∂E}{∂y_2} & ... & \frac{∂E}{∂y_n} \end{bmatrix}$$

$$\frac{∂E}{∂W} = \begin{bmatrix} \frac{∂E}{∂y_1} & \frac{∂E}{∂y_2} & ... & \frac{∂E}{∂y_n} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta W \end{bmatrix}$$

Note that each variable parameter $y_i$ contributes a term of the form

$$\frac{∂E}{∂y_i} \Delta y_i$$

Step-by-Step Derivation

The key to the derivation is Taylor series approximation:
The objective function for finding the best match is

$$w^T_j = e^{\text{ss image patch}}$$

The Lucas-Kanade algorithm minimizes the above objective function in a Gauss-Newton gradient descent non-linear optimization process.

$$E = \sum [I(W([x,y];P)) - T([x,y])]^2$$

$$-\sum [I(W([x,y];P)]) + \nabla I \frac{\partial W}{\partial P} \Delta P - T([x,y])^2$$

where

$$\nabla I = \frac{\partial I}{\partial P}$$

is the gradient of image I at W([x,y];P)

$$\frac{\partial W}{\partial P}$$

is the Jacobian of the warp

**Algorithm At a Glance**

**Algorithm Summary**

<table>
<thead>
<tr>
<th>Iterate</th>
<th>Source: “Lucas-Kanade 20 years on: A unifying framework” Baker and Matthews, IJCV 04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp I to obtain I(W([x,y];P))</td>
<td></td>
</tr>
<tr>
<td>Compute the error image T(x) - I(W([x,y];P))</td>
<td></td>
</tr>
<tr>
<td>Warp the gradient (\nabla I) with W([x,y];P)</td>
<td></td>
</tr>
<tr>
<td>Evaluate (\frac{\partial W}{\partial P}) at ([x,y];P)</td>
<td></td>
</tr>
<tr>
<td>Compute Hessian matrix (\nabla \nabla I^{\partial W/\partial P})</td>
<td></td>
</tr>
<tr>
<td>Compute steepest descent images (\nabla \nabla I^{\partial W/\partial P})</td>
<td></td>
</tr>
<tr>
<td>Compute (\sum (\nabla \nabla I^{\partial W/\partial P})(T(x,y) - I(W([x,y];P))))</td>
<td></td>
</tr>
<tr>
<td>Compute (\Delta P)</td>
<td></td>
</tr>
<tr>
<td>Update (P = P + \Delta P)</td>
<td></td>
</tr>
<tr>
<td>Until (\Delta P) magnitude is negligible</td>
<td></td>
</tr>
</tbody>
</table>

Source: “Lucas-Kanade 20 years on: A unifying framework” Baker and Matthews, IJCV 04
Approximating Normalized Correlation

Traditional LK is sensitive to changes in brightness (SSD makes a constant brightness assumption!)

We can remove some potential error sources (changes in illumination or camera gain) if we first normalize the template and the local search window by subtracting off mean and dividing by standard deviation.

MultiScale Gradient Descent

- Traditional LK is just refining a position estimate
- We must be close to the right answer to start with, and we are only guaranteed to descend to a local minimum in the SSD function (might not be best)
- To increase range of LK methods, we can apply them in a coarse to fine image pyramid, so at each level a small refinement is sufficient.

State of the Art Lucas Kanade Tracking

- Papers:
  - Original Paper [Baker and Matthews, CVPR, 2001]
  - Inverse Compositional Algorithm [Baker and Matthews, IJCV, 2004]
  - Application to AAMs [Matthews and Baker, IJCV, 2004]

Baker, Matthews, CMU

Using LK to Align the Background

General idea: We can also apply LK to align the whole image, which amounts to stabilizing the background of a moving camera.

\[
X = [a b o d e f g h]^T
\]
Recall: Projective Warping

Usually need at least a projective warp to align the image, because field of view is larger. Assumption: 1) background is approximately planar OR 2) camera motion is mostly rotational (small translation).

![Source Image](image1) ![Destination Image](image2)

Estimating a Homography

Matrix Form:

\[
\begin{bmatrix}
    x' \\
y' \\
1
\end{bmatrix} =
\begin{bmatrix}
    h_{11} & h_{12} & h_{13} \\
h_{21} & h_{22} & h_{23} \\
h_{31} & h_{32} & h_{33}
\end{bmatrix}
\begin{bmatrix}
    x \\
y \\
1
\end{bmatrix}
\]

Equations:

\[
x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} = W_x(x, y; h_{11}, ..., h_{33})
\]

\[
y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1} = W_y(x, y; h_{11}, ..., h_{33})
\]

Estimating a Homography

Actually, we need to enforce that there are only eight degrees of freedom. Is usually suffices to set \( h_{33} = 1 \).

Matrix Form:

\[
\begin{bmatrix}
    x' \\
y' \\
1
\end{bmatrix} =
\begin{bmatrix}
    h_{11} & h_{12} & h_{13} \\
h_{21} & h_{22} & h_{23} \\
h_{31} & h_{32} & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
y \\
1
\end{bmatrix}
\]

Equations:

\[
x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} = W_x(x, y; h_{11}, ..., h_{32})
\]

\[
y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1} = W_y(x, y; h_{11}, ..., h_{32})
\]

Degrees of Freedom?

There are 9 numbers \( h_{11}, ..., h_{33} \), so are there 9 DOF?
No. Note that we can multiply all \( h_{ij} \) by nonzero \( k \) without changing the equations:

\[
x' = \frac{kh_{11}x + kh_{12}y + kh_{13}}{kh_{31}x + kh_{32}y + kh_{33}} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}
\]

\[
y' = \frac{kh_{21}x + kh_{22}y + kh_{23}}{kh_{31}x + kh_{32}y + kh_{33}} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}
\]

Application: Stabilization

Given a sequence of video frames, warp them into a common image coordinate system.

This “stabilizes” the video to appear as if the camera is not moving.
Motivation for Stabilization

- Make it easier to distinguish object motion from camera motion
- Allows simpler object motion prediction models (e.g. constant velocity) to be more accurate.
- Easier for people to look at

Stabilization by Chaining

What if the reference image does not overlap with all the source images? As long as there are pairwise overlaps, we can chain (compose) pairwise homographies.

Not recommended for long sequences, as alignment errors accumulate over time.

Application: Mosaicing

Fixation via Mosaicing

incoming image is aligned to a background mosaic and adjustments to pan/tilt are made to keep a feature near center of the image.

Note on Background Stabilization

Homography assumes scene is roughly planar.

What if scene isn’t planar? Alignment will not be good if significant 3D relief

“parallax”
An alternate, feature-based approach

Idea: Sparse Feature Matching

General idea:
- Find corners in one image (because these are good areas to estimate displacement)
- Use translation-only LK to estimate displacement
- Fit higher-order parametric motion to the sparse displacements to estimate the warp

KLT Algorithm

Public-domain code by Stan Birchfield.
Available from
http://www.ces.clemson.edu/~stb/klt/

Tracking corner features through 2 or more frames

KLT Algorithm

1. Find corners in first image
2. Extract intensity patch around each corner
3. Use Lucas-Kanade algorithm to estimate constant displacement of pixels in patch

Niceties
1. Iteration and multi-resolution to handle large motions
2. Subpixel displacement estimates (bilinear interp warp)
3. Can track feature through a whole sequence of frames
4. Ability to add new features as old features get “lost”
We now have a set of point correspondences \((x, y) \rightarrow (x', y')\)

Assume related by homography:
\[
\begin{align*}
x' &= \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \\
y' &= \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}
\end{align*}
\]

Solve for \(h_{11}, \ldots, h_{32}\) using least squares.

### L.S. using Algebraic Distance

\[
\begin{align*}
x' &= \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \\
y' &= \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}
\end{align*}
\]

Multiplying through by denominator
\[
\begin{align*}
(h_{11}x + h_{12}y + 1)x' &= h_{11}x + h_{12}y + h_{13} \\
(h_{21}x + h_{22}y + h_{23})y' &= h_{21}x + h_{22}y + h_{23}
\end{align*}
\]

Rearrange
\[
\begin{align*}
h_{11}x + h_{12}y + h_{13} - h_{13}xx' - h_{12}yy' &= x' \\
h_{21}x + h_{22}y + h_{23} - h_{13}xy' - h_{23}yy' &= y'
\end{align*}
\]

### Caution: Numeric Conditioning

R. Hartley: “In Defense of the Eight Point Algorithm”

Observation: Linear estimation of projective transformation parameters from point correspondences often suffer from poor “conditioning” of the matrices involved. This means the solution is sensitive to noise in the points (even if there are no outliers).

To get better answers, precondition the matrices by performing a normalization of each point set by:
- translating center of mass to the origin
- scaling so that average distance of points from origin is \(\sqrt{2}\).
- do this normalization to each point set independently

Read the paper if you care (it is a nice paper). But usually in matlab, using double float, it doesn’t matter.