Efficient & Robust LK for Mobile Vision

Instructor - Simon Lucey

16-423 - Designing Computer Vision Apps
Today

• Review - LK Algorithm
• Inverse Composition
• Robust Extensions
Lucas & Kanade (1981) realized this and proposed a method for estimating warp displacement using the principles of *gradients* and *spatial coherence*.

Technique applies Taylor series approximation to any spatially coherent area governed by the warp \( \mathcal{W}(x; p) \).

\[
\mathcal{I}(p + \Delta p) \approx \mathcal{I}(p) + \frac{\partial \mathcal{I}(p)}{\partial p^T} \Delta p
\]
Lucas & Kanade (1981) realized this and proposed a method for estimating warp displacement using the principles of \textit{gradients} and \textit{spatial coherence}.

Technique applies Taylor series approximation to any spatially coherent area governed by the warp $\mathcal{W}(x; p)$.

$$
\mathcal{I}(p + \Delta p) \approx \mathcal{I}(p) + \frac{\partial \mathcal{I}(p)}{\partial p^T} \Delta p
$$
Review - LK Algorithm

- Lucas & Kanade (1981) realized this and proposed a method for estimating warp displacement using the principles of *gradients* and *spatial coherence*.

- Technique applies Taylor series approximation to any spatially coherent area governed by the warp $\mathcal{W}(x; p)$.

\[
 I(p + \Delta p) \approx I(p) + \frac{\partial I(p)}{\partial p^T} \Delta p
\]

"We consider this image to always be static...."
Lucas & Kanade (1981) realized this and proposed a method for estimating warp displacement using the principles of **gradients** and **spatial coherence**.

Technique applies Taylor series approximation to any spatially coherent area governed by the warp $\mathcal{W}(x; p)$.

$$
\mathcal{I}(p + \Delta p) \approx \mathcal{I}(p) + \frac{\partial \mathcal{I}(p)}{\partial p_T} \Delta p
$$
• Lucas & Kanade (1981) realized this and proposed a method for estimating warp displacement using the principles of \textit{gradients} and \textit{spatial coherence}.

• Technique applies Taylor series approximation to any spatially coherent area governed by the warp $\mathcal{W}(\mathbf{x}; \mathbf{p})$.

\[
\mathcal{I}(\mathbf{p} + \Delta \mathbf{p}) \approx \mathcal{I}(\mathbf{p}) + \frac{\partial \mathcal{I}(\mathbf{p})}{\partial \mathbf{p}^T} \Delta \mathbf{p}
\]

\[
\frac{\partial \mathcal{I}(\mathbf{p})}{\partial \mathbf{p}^T} = \begin{bmatrix}
\frac{\partial \mathcal{I}(\mathbf{x}'_1)}{\partial \mathbf{x}'_1^T} & \cdots & \mathbf{0}^T \\
\vdots & \ddots & \vdots \\
\mathbf{0}^T & \cdots & \frac{\partial \mathcal{I}(\mathbf{x}'_N)}{\partial \mathbf{x}'_N^T}
\end{bmatrix} \begin{bmatrix}
\frac{\partial \mathcal{W}(\mathbf{x}_1; \mathbf{p})}{\partial \mathbf{p}^T} \\
\vdots \\
\frac{\partial \mathcal{W}(\mathbf{x}_N; \mathbf{p})}{\partial \mathbf{p}^T}
\end{bmatrix}
\]

\[
\mathbf{x}' = \mathcal{W}(\mathbf{x}; \mathbf{p})
\]
Reminder - Template Coordinates

$\mathcal{W}(x_1; p)$

$\mathcal{W}(x_1; 0)$

$I$

“Source Image”

$T$

“Template”
Reminder - Template Coordinates

\[ x'_{1} \quad I \quad \text{“Source Image”} \]

\[ x_{1} \quad T \quad \text{“Template”} \]
\[
\frac{\partial \mathcal{I}(\mathbf{p})}{\partial \mathbf{p}^T} = \begin{bmatrix}
\frac{\partial \mathcal{I}(\mathbf{x}'_1)}{\partial \mathbf{x}'_1^T} & \ldots & 0^T \\
\vdots & \ddots & \vdots \\
0^T & \ldots & \frac{\partial \mathcal{I}(\mathbf{x}'_N)}{\partial \mathbf{x}'_N^T}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial \mathcal{W}(\mathbf{x}_1; \mathbf{p})}{\partial \mathbf{p}^T} \\
\vdots \\
\frac{\partial \mathcal{W}(\mathbf{x}_N; \mathbf{p})}{\partial \mathbf{p}^T}
\end{bmatrix}
\]

\[\nabla_x \mathcal{I}\]

\[\nabla_y \mathcal{I}\]
Which Way?

**Step 1: Warp Image**

\[ \mathcal{I} \rightarrow \mathcal{I}(p) \]

**Step 2: Estimate Gradients**

\[ \nabla_x \mathcal{I}(p) \quad \nabla_y \mathcal{I}(p) \]
Which Way?

**Step 1: Warp Image**

\[ \mathcal{I} \]

\[ \mathcal{I}(p) \]

**Step 2: Estimate Gradients**

\[ \frac{\partial \mathcal{I}(x')}{\partial x} \]
Which Way?

**Step 1: Warp Image**

\[ \mathcal{I} \]  
\[ \mathcal{I}(p) \]

**Step 2: Estimate Gradients**

\[ \mathcal{I}(p) \]

\[ \frac{\partial \mathcal{I}(x')}{\partial x} \]
Which Way?

**Step 1: Estimate Gradients**

\[ \nabla_x I \]

"Horizontal"

\[ \nabla_y I \]

"Vertical"

**Step 2: Warp Gradients**

\[ \nabla_x I(p) \]

\[ \nabla_y I(p) \]
Which Way?

**Step 1: Estimate Gradients**

$I$

* "Horizontal"

* "Vertical"

$\nabla_x I$

$\nabla_y I$

**Step 2: Warp Gradients**

$\frac{\partial I(x')}{\partial x'}$
\[ \frac{\partial \mathcal{I}(p)}{\partial p^T} = \begin{bmatrix} \frac{\partial \mathcal{I}(x'_1)}{\partial x'_1^T} & \cdots & 0^T \\ \vdots & \ddots & \vdots \\ 0^T & \cdots & \frac{\partial \mathcal{I}(x'_N)}{\partial x'_N^T} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{W}(x_1; p)}{\partial p^T} \\ \vdots \\ \frac{\partial \mathcal{W}(x_N; p)}{\partial p^T} \end{bmatrix} \]
For an affine warp,

\[ W(x; p) = \begin{bmatrix} 1 - p_1 & p_2 & p_3 \\ p_4 & 1 - p_5 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \]

\[ \frac{\partial W(x; p)}{\partial p^T} = \begin{bmatrix} -x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & -y & 1 \end{bmatrix} \]
Review - LK Algorithm

- Lucas & Kanade (1981) realized this and proposed a method for estimating warp displacement using the principles of *gradients* and *spatial coherence*.
- Technique applies Taylor series approximation to any spatially coherent area governed by the warp $\mathcal{W}(x; p)$.

$$\mathcal{I}(p + \Delta p) \approx \mathcal{I}(p) + \frac{\partial \mathcal{I}(p)^T}{\partial p} \Delta p$$

- $N = \text{number of pixels}$
- $P = \text{number of warp parameters}$
Review - LK Algorithm

- Often just refer to,
  \[ J_\mathcal{I} = \frac{\partial \mathcal{I}(p)}{\partial p^T} \]
  as the "Jacobian" matrix.

- Also refer to,
  \[ H_\mathcal{I} = J_\mathcal{I}^T J_\mathcal{I} \]
  as the "pseudo-Hessian".

- Finally, we can refer to,
  \[ \mathcal{T}(0) = \mathcal{I}(p + \Delta p) \]
  as the "template".
LK Algorithm

- Actual algorithm is just the application of the following steps,

**Step 1:**

\[
\Delta p = H_{T}^{-1} J_{I}^{T} [T(0) - I(p)]
\]

**Step 2:**

\[
p \leftarrow p + \Delta p
\]

keep applying steps until \( \Delta p \) converges.
Examples of LK Alignment
Examples of LK Alignment
Gauss-Newton Algorithm

• LK is essentially an application of the Gauss-Newton algorithm,

\[
\arg \min_x \| y - F(x) \|^2_2
\]

s.t. \( F : \mathbb{R}^N \to \mathbb{R}^M \)

**Step 1:**

\[
\arg \min_{\Delta x} \| y - F(x) - \frac{\partial F(x)}{\partial x^T} \Delta x \|^2_2
\]

**Step 2:**

\[ x \leftarrow x + \Delta x \]

keep applying steps until \( \Delta x \) converges.
The optimization employed with the LK algorithm can be interpreted as Gauss-Newton optimization.

Other non-linear least-squares optimization strategies have been investigated (Baker et al. 2003)

- Levenberg-Marquadt
- Newton
- Steepest-Descent

Gauss-Newton in empirical evaluations has appeared to be the most robust (Baker et al.).
• Initial warp estimate has to be suitably close to the ground-truth for gradient-search methods to work.
• Kind of like a black hole’s event horizon.
• You got to be inside it to be sucked in!
Expanding the Event Horizon

- Best strategy is to expand the neighborhood $\mathbb{N}$ across which gradients are estimated.
- Simplest to do this in practice with a blur.
- Often apply what is known as “coarse-to-fine” alignment.
Questions

• Why is LK sensitive to the initial guess?
• Why can’t you perform the optimization in a single shot?
Today

• Review - LK Algorithm
• Inverse Composition
• Robust Extensions
Efficient search is essential on mobile and desktop!!
Computation Concerns

- Unfortunately, the LK algorithm can be computationally expensive.
  - Requires the re-computation of the Jacobian matrix at each iteration.
  - With the additional inversion of the Hessian matrix at each iteration.

\[
\mathbf{J}_I = \begin{bmatrix}
\frac{\partial \mathcal{I}(\mathbf{x}'_1)}{\partial \mathbf{x}'_1^T} & \cdots & \mathbf{0}^T \\
\vdots & \ddots & \vdots \\
\mathbf{0}^T & \cdots & \frac{\partial \mathcal{I}(\mathbf{x}'_N)}{\partial \mathbf{x}'_N^T}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial \mathcal{W}(\mathbf{x}_1; \mathbf{p})}{\partial \mathbf{p}^T} \\
\vdots \\
\frac{\partial \mathcal{W}(\mathbf{x}_N; \mathbf{p})}{\partial \mathbf{p}^T}
\end{bmatrix}
\]

- With the additional inversion of the Hessian matrix at each iteration.

\[
\mathbf{H}_I = \mathbf{J}_I^T \mathbf{J}_I
\]

“Is there any way we can pre-compute any of this?”
Linearizing the Template?

\[ I(p + \Delta p) \approx I(p) + \frac{\partial I(p)^T}{\partial p} \Delta p \]
Linearizing the Template?

\[ T(0) \approx I(p) + \frac{\partial I(p)}{\partial p^T} \Delta p \]

“template”
Linearizing the Template?

\[ \mathcal{T}(0) \approx \mathcal{I}(\mathbf{p}) + \frac{\partial \mathcal{I}(\mathbf{p})}{\partial \mathbf{p}^T} \Delta \mathbf{p} \]

“Why is this useful if the template must be static?”
Additive Warps

\[ \mathcal{W}(x_1; p) \rightarrow \mathcal{W}(x_2; p + \Delta p) \rightarrow \mathcal{W}(x_3; p + \Delta p) \]
Additive Warps

Classically we seek an additive warp update,

\[
\arg \min_{\Delta M} \sum_{n=1}^{N} \|\tilde{x}'_n - (M + \Delta M)\tilde{x}_n\|_2^2
\]

\[
M \rightarrow M + \Delta M \quad \text{“Warp Update”}
\]

\[
M = \begin{bmatrix}
1 - p_1 & p_2 & p_3 \\
p_4 & 1 - p_5 & p_6 \\
0 & 0 & 1
\end{bmatrix} \quad \text{“Affine warp”}
\]

\[
\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \quad \text{“Homogeneous coordinates”}
\]
Additive Warps

Classically we seek an additive warp update,

$$\arg \min_{\Delta M} \sum_{n=1}^{N} \left\| \tilde{x}'_n - (M + \Delta M)\tilde{x}_n \right\|^2_2$$

$M = \begin{bmatrix} 1 - p_1 & p_2 & p_3 \\ p_4 & 1 - p_5 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$

$\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$
Additive Warps

- Classically we seek an additive warp update,

$$\arg \min_{\Delta M} \sum_{n=1}^{N} \left\| \tilde{x}'_n - (M + \Delta M)\tilde{x}_n \right\|_2^2$$

$$\mathbf{p} \rightarrow \mathbf{p} + \Delta \mathbf{p}$$  “Warp Update”

$$\mathbf{M} = \begin{bmatrix} 1 - p_1 & p_2 & p_3 \\ p_4 & 1 - p_5 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$  “Affine warp”

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$  “Homogeneous coordinates”
Forwards Compositional Warps

• Alternatively we can seek a compositional update,

\[
\arg \min_{\Delta M} \sum_{n=1}^{N} \left\| \tilde{x}'_n - (\Delta M) M \tilde{x}_n \right\|_2^2
\]

\[
M \rightarrow (\Delta M) M
\]

\[
M = \begin{bmatrix}
1 - p_1 & p_2 & p_3 \\
p_4 & 1 - p_5 & p_6 \\
0 & 0 & 1
\end{bmatrix}
\]

“Affine warp”
Forwards Compositional Warps

- Alternatively we can seek a compositional update,

\[
\arg\min_{\Delta M} \sum_{n=1}^{N} \| \tilde{x}_n' - (\Delta M) M \tilde{x}_n \|^2 \]

\[
p \rightarrow p \circ \Delta p
\]

\[
M = \begin{bmatrix}
1 - p_1 & p_2 & p_3 \\
p_4 & 1 - p_5 & p_6 \\
0 & 0 & 1
\end{bmatrix}
\]

“Affine warp”
Inverse Compositional Warps

• Interestingly, we can also estimate the update on one side

$$\text{arg min}_{\Delta M} \sum_{n=1}^{N} ||\Delta M \tilde{x}'_n - M \tilde{x}_n||^2_2$$

but, then apply it to the other

$$M \rightarrow (\Delta M)^{-1} M \quad \text{“Warp Update”}$$

$$M = \begin{bmatrix} 1 - p_1 & p_2 & p_3 \\ p_4 & 1 - p_5 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{“Affine warp”}$$
Inverse Compositional Warps

• Interestingly, we can also estimate the update on one side

\[
\arg \min_{\Delta M} \sum_{n=1}^{N} \| \Delta M \tilde{x}'_n - M \tilde{x}_n \|_2^2
\]

but, then apply it to the other

\[
p \rightarrow p \circ^{-1} \Delta p
\]

“Warp Update”

\[
M = \begin{bmatrix}
1 - p_1 & p_2 & p_3 \\
p_4 & 1 - p_5 & p_6 \\
0 & 0 & 1
\end{bmatrix}
\]

“Affine warp”
Reminder: Linearizing the Template?

\[ T(0) \approx I(p) + \frac{\partial I(p)}{\partial p^T} \Delta p \]

"template"

\[ T(0) + \frac{\partial T(0)}{\partial p^T} \Delta p \approx I(p) \]
Inverse Composition Algorithm

• Actual algorithm is just the application of the following steps,

   **Step 1:**
   
   \[
   \arg \min_{\Delta p} \left\| \mathcal{T}(0) + \frac{\partial \mathcal{T}(0)}{\partial p^T} \Delta p - \mathcal{I}(p) \right\|^2_2
   \]

   **Step 2:**
   
   \[ p \rightarrow p \odot^{-1} \Delta p \]

   keep applying steps until \( \Delta p \) converges.
Inverse Composition Algorithm

- Actual algorithm is just the application of the following steps,

**Step 1:**

\[ \Delta p = H^{-1}_T J^T_T [\mathcal{I}(p) - \mathcal{T}(0)] \]

**Step 2:**

\[ p \rightarrow p \circ^{-1} \Delta p \]

keep applying steps until \( \Delta p \) converges.

\[ J_T = \frac{\partial \mathcal{T}(0)}{\partial p^T} \quad H_T = J^T_T J_T \]
Inverse Composition Algorithm

• Actual algorithm is just the application of the following steps,

  **Step 1:**
  \[
  \Delta p = H^{-1}_T J^T_J \left[ I(p) - \mathcal{T}(0) \right]
  \]

  “Static”

  **Step 2:**
  \[
  p \rightarrow p \circ^{-1} \Delta p \quad \text{“Inverse Composition”}
  \]

  keep applying steps until \( \Delta p \) converges.

  \[
  J_T = \frac{\partial \mathcal{T}(0)}{\partial p^T} \quad \text{H}_T = J^T_T J_T
  \]
**Linearizing the Template**

\[
\frac{\partial \mathcal{I}(p)}{\partial p^T} = \begin{bmatrix}
\frac{\partial \mathcal{I}\{\mathcal{W}(x_1;p)\}}{\partial \mathcal{W}(x_1;p)^T} & \cdots & 0^T \\
\vdots & \ddots & \vdots \\
0^T & \cdots & \frac{\partial \mathcal{I}\{\mathcal{W}(x_N;p)\}}{\partial \mathcal{W}(x_N;p)^T}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial \mathcal{W}(x_1;p)}{\partial p^T} \\
\vdots \\
\frac{\partial \mathcal{W}(x_N;p)}{\partial p^T}
\end{bmatrix}
\]

“Constantly changing”

\[
\frac{\partial \mathcal{T}(0)}{\partial p^T} = \begin{bmatrix}
\frac{\partial \mathcal{T}(x_1)}{\partial x_1^T} & \cdots & 0^T \\
\vdots & \ddots & \vdots \\
0^T & \cdots & \frac{\partial \mathcal{T}(x_N)}{\partial x_N^T}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial \mathcal{W}(x_1;0)}{\partial p^T} \\
\vdots \\
\frac{\partial \mathcal{W}(x_N;0)}{\partial p^T}
\end{bmatrix}
\]

“Static”
Linearizing the Template

\[
\frac{\partial I(p)}{\partial p^T} = \begin{bmatrix}
\frac{\partial I\{W(x_1;p)\}}{\partial W(x_1;p)^T} & \cdots & 0^T \\
\vdots & \ddots & \vdots \\
0^T & \cdots & \frac{\partial I\{W(x_N;p)\}}{\partial W(x_N;p)^T}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial W(x_1;p)}{\partial p^T} \\
\vdots \\
\frac{\partial W(x_N;p)}{\partial p^T}
\end{bmatrix}
\]

“Constantly changing”

\[
\frac{\partial T(0)}{\partial p^T} = \begin{bmatrix}
\frac{\partial T(x_1)}{\partial x_1^T} & \cdots & 0^T \\
\vdots & \ddots & \vdots \\
0^T & \cdots & \frac{\partial T(x_N)}{\partial x_N^T}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial W(x_1;0)}{\partial p^T} \\
\vdots \\
\frac{\partial W(x_N;0)}{\partial p^T}
\end{bmatrix}
\]

“Static”
Rules for Compositional Warps

• Not just applicable to affine warps, can be applied to any set of warps that:
  • have an identity warp (i.e. $\mathcal{W}(x; 0) = x$)
  • set of warps must be closed under composition,

\[
\mathcal{W}(\mathcal{W}(x; p_1); p_2) \rightarrow \mathcal{W}(x; p_2 \circ p_1)
\]

for example a homography,

\[
\begin{pmatrix}
1 - p_1^{(a)} & p_2^{(a)} & p_3^{(a)} \\
p_4^{(a)} & 1 - p_5^{(a)} & p_6^{(a)} \\
p_7^{(a)} & p_8^{(a)} & 1
\end{pmatrix}
\begin{pmatrix}
1 - p_1^{(b)} & p_2^{(b)} & p_3^{(b)} \\
p_4^{(b)} & 1 - p_5^{(b)} & p_6^{(b)} \\
p_7^{(b)} & p_8^{(b)} & 1
\end{pmatrix}
= \lambda
\begin{pmatrix}
1 - p_1^{(c)} & p_2^{(c)} & p_3^{(c)} \\
p_4^{(c)} & 1 - p_5^{(c)} & p_6^{(c)} \\
p_7^{(c)} & p_8^{(c)} & 1
\end{pmatrix}
\]
## Inverse Compositional Performance

<table>
<thead>
<tr>
<th>Template area</th>
<th>Intensity (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>75 × 57</td>
<td>650</td>
</tr>
<tr>
<td>150 × 115</td>
<td>360</td>
</tr>
<tr>
<td>300 × 230</td>
<td>140</td>
</tr>
<tr>
<td>640 × 460</td>
<td>45</td>
</tr>
</tbody>
</table>

Planar template tracking runtime in frames per second on single core Intel Core i7 2.8 Ghz.
Today

• Review - LK Algorithm
• Inverse Composition
• Robust Extensions
Robust Error Functions

- Least squares criterion is not robust to outliers (e.g. occlusion, illumination)

- For example, the two outliers here cause the fitted line to be quite wrong.

- Robust error functions can be helpful here.
Robust Error Functions

• Least squares criterion is not robust to outliers (e.g. occlusion, illumination)

• For example, the two outliers here cause the fitted line to be quite wrong.

• Robust error functions can be helpful here.
Robust Error Functions

\[
\arg \min_{\Delta p} \eta \{ \mathcal{T}(0) + \frac{\partial \mathcal{T}(0)}{\partial p^T} \Delta p - \mathcal{I}(p) \} 
\]

for example,

\[
\eta \{ x \} = ||x||_1 \rightarrow L_1 \text{ error}
\]
Robust Error Functions

• Computational inefficiencies creep into inverse composition when one departs from the classical L2 distance.
• Much better results for fast tracking have been achieved recently using robust representations (e.g. Bristow et al.).
Robust Representations

1. Compute image gradients
2. Pool into local histograms
3. Concatenate histograms
4. Normalize histograms

\[ \psi \rightarrow \mathbb{R}^N : \mathbb{R}^{N \times K} \]
Actual algorithm is just the application of the following steps,

\[
\begin{align*}
\text{Step 1:} & \quad \arg\min_{\Delta p} \| \psi \{ T(0) \} + \frac{\partial \psi \{ T(0) \}}{\partial p^T} \Delta p - \psi \{ I(p) \} \|^2_2 \\
\text{Step 2:} & \quad p \rightarrow p \circ^{-1} \Delta p
\end{align*}
\]

keep applying steps until $\Delta p$ converges.
### Descriptor Performance

**“BitPlanes”**

**“Raw Pixels”**

<table>
<thead>
<tr>
<th>Template area</th>
<th>Intensity</th>
<th>BitPlanes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$75 \times 57$</td>
<td>650</td>
<td>460</td>
</tr>
<tr>
<td>$150 \times 115$</td>
<td>360</td>
<td>170</td>
</tr>
<tr>
<td>$300 \times 230$</td>
<td>140</td>
<td>90</td>
</tr>
<tr>
<td>$640 \times 460$</td>
<td>45</td>
<td>35</td>
</tr>
</tbody>
</table>

Planar template tracking runtime in frames per second on single core Intel Core i7 2.8 Ghz.
More to read...