Global Flow

Anandan

Affine

Global Motion
- Estimate motion using all pixels in the image.
- Parametric flow gives an equation, which describes optical flow for each pixel.
  - Affine
  - Projective
- Global motion can be used to
  - generate mosaics
  - Object-based segmentation

Affine

\[
\begin{align*}
\begin{bmatrix}
u(x',y') \\ v(x',y')
\end{bmatrix} &= \begin{bmatrix}
a_1 & a_2 \\ a_3 & a_4
\end{bmatrix}
\begin{bmatrix}
x \\ y
\end{bmatrix} + \begin{bmatrix}
b_1 \\ b_2
\end{bmatrix}
\end{align*}
\]

Spatial Transformations
- Translation
- Rotation
- Shear
- Affine
- Rigid (rotation and translation)
Anandan

**Optical flow constraint**

\[ f_x u + f_y v = -f_t \]

\[ E(u) = \sum_{(x,y)} (f_x + f_y' u)^2 \]

\[ E(a) = \sum_{(x,y)} (f_x + f_y' X(x)a)^2 \]

(a) Derive this

\[ E(\delta a) = \sum_{(x,y)} (f_x + f_y' Xa)^2 \]

**Linear system**

\[ Ax = b \]

Basic Components

- Pyramid construction
- Motion estimation
- Image warping
- Coarse-to-fine refinement

Image Warping

- Warping an image \( f \) into image \( h \) using some transformation \( g \), involves mapping intensity at each pixel \((x,y)\) in image \( f \) to a pixel \((g(x),g(y))\) image \( h \) such that

\[ (x',y') = (g(x),g(y)) \]

- In case of affine transformation, \( (x',y') \) is transformed to \( x'=Ax+b \)

\[ f(x,y,t-1) \]

\[ f(\cdot) \]

\[ f(\cdot) \]

\[ f(\cdot) \]
Image Warping

\[ X' = X - U = X - (AX + b) \]
\[ f(X', t-1) \]
\[ f(X, t) \]
\[ (I - A)^{-1}(X' + b) = X'' \]
\[ (A')^{-1}(X' + b) = X'' \]

• How about values in \( X'' = (x'', y'') \) are not integer.
• But image is sampled only at integer rows and columns
  • Instead of converting \( X' \) to \( X'' \) and copying \( f(X', t-1) \) at \( f(X'', t-1) \), we can convert integer values \( X' \) to \( X'' \) and copy \( f(X', t-1) \) at \( f(X'', t-1) \).

Image Warping

\[ X' = X - U = X - (AX + b) \]
\[ X' = (I - A)X - b \]
\[ X' + b = AX \]
\[ (A')^{-1}(X' + b) = X \]
\[ (A')^{-1}(X' + b) = X'' \]
\[ X' \rightarrow X'' \]

• But how about the values in \( X'' = (x'', y'') \) are not integer.
• Perform bilinear interpolation to compute \( f(X', t-1) \) at non-integer values.

Image Warping

\[ (A')^{-1}(X' + b) = X'' \]
\[ (X' + b) = (A')X'' \]
\[ X' = (A')X'' - b \]
\[ X'' \rightarrow X' \]
Warping

Video Mosaic

Sprite

Szelski

Projective
Projective

\[
\begin{align*}
    f(x',t-1) &= f(x,t) \\
    x' &= \frac{a_2x + a_3y + b_1}{c_1x + c_2y + 1} \\
    y' &= \frac{a_3x + a_4y + b_2}{c_1x + c_2y + 1}
\end{align*}
\]

Szeliski

Motion Vector:

\[
m = [a_1, a_2, a_3, a_4, b_1, b_2, c_1, c_2]'
\]

\[
\begin{align*}
    \Delta m &= (A + \lambda I)^{-1} b \\
    a_{ij} &= \sum \frac{\partial e_i}{\partial m_j} \quad b_i = -\sum \frac{\partial e_i}{\partial m_i}
\end{align*}
\]

Szeliski (Levenberg-Marquadt)

Approximation of Hessian (J^T J, Jacobian)

\[
A = J^T J
\]

Approximation of Hessian

\[
a_{ij} = \sum \frac{\partial e_i}{\partial m_j} \quad A \text{ Matrix}
\]
Partial Derivatives wrt image coordinates

$$E = \sum [f(x', y') - f(x, y)]^2 = \sum e^2$$

$$\frac{\partial e}{\partial x'} = f_x$$

$$\frac{\partial e}{\partial y'} = f_y$$

Partial Derivatives wrt motion parameters

$$\frac{\partial e}{\partial a_i} = \frac{\partial e}{\partial a_i} + \frac{\partial e}{\partial a_i}$$

$$\frac{\partial e}{\partial b_i} = \frac{\partial e}{\partial b_i} + \frac{\partial e}{\partial b_i}$$

Szeliski (Levenberg-Marquadt)

- Start with some initial value of $m$, and $\lambda=.001$

- For each pixel $I$ at $(x_i, y_i)$

  - Compute $(x', y')$ using projective transform.

  - Compute $e = f(x', y') - f(x, y)$

  - Compute $\frac{\partial e}{\partial m_i} = \frac{\partial e}{\partial x'} \frac{\partial x'}{\partial m_i} + \frac{\partial e}{\partial y'} \frac{\partial y'}{\partial m_i}$
Szeliski (Levenberg-Marquadt)

- Compute $A$ and $b$

- Solve system $(A - \lambda I)\Delta m = b$

- Update $m^{i+1} = m^i + \Delta m$

Szeliski (Levenberg-Marquadt)

- Check if error has decreased, if not increase $\lambda$ by a factor of 10 and compute a new $\Delta m$

- If error has decreased, decrease $\lambda$ by a factor of 10 and compute a new $\Delta m$

- Continue iteration until error is below threshold.