THE GEOMETRY OF MULTIPLE VIEWS
Lecture 11

Siamak Khatibi
BELEKINGE INSTITUTE OF TECHNOLOGY

Reading: Chapter 10.
THE GEOMETRY OF MULTIPLE VIEWS

• Epipolar Geometry
  • The Essential Matrix
  • The Fundamental Matrix
**Review: Intrinsic Camera Parameters**

Image plane

\[ i = k_u I \]
\[ j = k_v J \]

Focal plane

\[ (u^{(c)}, v^{(c)}, f) \]
\[ (u^{(r)}, v^{(r)}) \]

\[
\begin{bmatrix}
U^{(new)}
\end{bmatrix} = \begin{bmatrix}
-f_u & 0 & u_0 & 0 \\
0 & -f_v & v_0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} \begin{bmatrix}
X^{(c)} \\
Y^{(c)} \\
Z^{(c)} \\
1
\end{bmatrix}
\]

\[ f_u = fk_u \]
\[ f_v = fk_v \]
**Review: Extrinsic Parameters**

By Rigid Body Transformation:

\[
\begin{bmatrix}
X^{(C)} \\
Y^{(C)} \\
Z^{(C)} \\
1
\end{bmatrix} =
\begin{bmatrix}
R_{3\times3} & T_{3\times1} \\
0_{1\times3} & 1
\end{bmatrix}
\begin{bmatrix}
X^{(W)} \\
Y^{(W)} \\
Z^{(W)} \\
1
\end{bmatrix} \Rightarrow M^{(C)} = DM^{(W)}
\]
Multi-View Geometry

Relates

• 3D World Points
• Camera Centers
• Camera Orientations
Multi-View Geometry

Relates

- 3D World Points
- Camera Centers
- Camera Orientations
- Camera Intrinsic Parameters
- Image Points
Stereo

scene point

image plane

optical center
Stereo

- Basic Principle: Triangulation
  - Gives reconstruction as intersection of two rays
  - Requires
    - calibration
    - point correspondence
Given p in left image, where can the corresponding point p’ in right image be?
Epipolar Geometry

- Epipolar Plane
- Epipoles
- Epipolar Lines
- Baseline
**Stereo Constraints**

- **Image plane**
- **Focal plane**
- **Epipolar Line**
- **Epipole**
Epipolar Constraint

- Potential matches for \( p \) have to lie on the corresponding epipolar line \( l' \).

- Potential matches for \( p' \) have to lie on the corresponding epipolar line \( l \).
From Geometry to Algebra

**Figure 11.1:** Epipolar geometry: the point $P$, the optical centers $O$ and $O'$ of the two cameras, and the two images $p$ and $p'$ of $P$ all lie in the same plane.
The epipolar constraint: these vectors are coplanar:

\[ \overrightarrow{OP} \cdot [\overrightarrow{OO'} \times \overrightarrow{O'p'}] = 0 \]
\[ \overrightarrow{OP} \cdot [\overrightarrow{OO'} \times \overrightarrow{O'P'}] = 0 \]

\[ p \cdot [t \times (Rp')] = 0 \]

Linear Constraint:
Should be able to express as matrix multiplication.
Review: Matrix Form of Cross Product

The vector cross product also acts on two vectors and returns a third vector. Geometrically, this new vector is constructed such that its projection onto either of the two input vectors is zero.

\[ \vec{a} \times \vec{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix} \]

\[ \vec{a} \times \vec{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \vec{c} \quad \vec{a} \cdot \vec{c} = 0 \quad \vec{b} \cdot \vec{c} = 0 \]
Review: Matrix Form of Cross Product

\[ \vec{a} \times \vec{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \vec{c} \]

\[ \vec{a} \cdot \vec{c} = 0 \]

\[ \vec{b} \cdot \vec{c} = 0 \]

\[ [a_x] = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \]

\[ \vec{a} \times \vec{b} = [a_x] \vec{b} \]
Matrix Form

\[ p \cdot \left[ t \times (\mathcal{R}p') \right] = 0 \]

\[ p^T [t_x] \mathcal{R} p' = 0 \]

\[ \vec{a} \times \vec{b} = [a_x] \vec{b} \]

\[ \varepsilon = [t_x] \mathcal{R} \]

\[ p^T \varepsilon p' = 0 \]
Epipolar Constraint: Calibrated Case

\[
\overrightarrow{O_p} \cdot [\overrightarrow{OO'} \times \overrightarrow{O'p'}] = 0 \quad \Rightarrow \quad \mathbf{p} \cdot [\mathbf{t} \times (\mathbf{R} \mathbf{p}')] = 0 \quad \text{with} \quad \begin{cases} \mathbf{p} = (u, v, 1)^T \\ \mathbf{p}' = (u', v', 1)^T \\ \mathbf{M} = (\mathbf{I} \quad \mathbf{0}) \\ \mathbf{M}' = (\mathbf{R}^T, -\mathbf{R}^T \mathbf{t}) \end{cases}
\]

Essential Matrix
(Longuet-Higgins, 1981)

\[
\mathbf{p}^T \mathbf{E} \mathbf{p}' = 0 \quad \text{with} \quad \mathbf{E} = [\mathbf{t} \times] \mathbf{R}
\]
The Essential Matrix

\( \mathcal{E}p' \) is the epipolar line corresponding to \( p' \) in the left camera.

\[ au + bv + c = 0 \]

\[ p = (u, v, 1)^T \]

\[ l = (a, b, c)^T \]

\[ l \cdot p = 0 \]

\[ \mathcal{E}p' \cdot p = 0 \]

\[ p^T \mathcal{E}p' = 0 \]

Similarly \( \mathcal{E}p^T \) is the epipolar line corresponding to \( p \) in the right camera.
The Essential Matrix

\[ \mathcal{E} e' = [t_x] R e' = 0 \]

Similarly,

\[ \mathcal{E}^T e = R^T [t_x]^T e = -R^T [t_x] e = 0 \]

Essential Matrix is singular with rank 2
Properties of the Essential Matrix

- $\mathcal{E} p'$ is the epipolar line associated with $p'$.
- $\mathcal{E}^T p$ is the epipolar line associated with $p$.
- $\mathcal{E} e'=0$ and $\mathcal{E}^T e=0$.
- $\mathcal{E}$ is singular.
- $\mathcal{E}$ has two equal non-zero singular values (Huang and Faugeras, 1989).

$p^T \mathcal{E} p' = 0$ with $\mathcal{E} = [t_x] \mathcal{R}$
The Essential Matrix

- Based on the Relative Geometry of the Cameras
- Assumes Cameras are calibrated (i.e., intrinsic parameters are known)
- Relates image of point in one camera to a second camera (points in camera coordinate system).
- Is defined up to scale
- 5 independent parameters
Fundamental Matrix

\( p^T \mathcal{E} p' = 0 \) \( p \) and \( p' \) are in camera coordinate system

If \( u \) and \( u' \) are corresponding image coordinates then we have

\[
\begin{align*}
  u &= P_1 p \\
  u' &= P_2 p' \\
  p &= P_1^{-1} u \\
  p' &= P_2^{-1} u'
\end{align*}
\]

\[
\begin{align*}
  u^T P_1^{-T} \mathcal{E} P_2^{-1} u' &= 0 \\
  \Rightarrow u^T F u' &= 0
\end{align*}
\]

\[ F = P_1^{-T} \mathcal{E} P_2^{-1} \]

Fundamental Matrix
Fundamental Matrix

\[ u^T F u' = 0 \]

\[ F = P_1^{−T} \mathcal{E} P_2^{-1} \]

Fundamental Matrix is singular with rank 2

In principal F has 7 parameters up to scale and can be estimated from 7 point correspondences

Direct Simpler Method requires 8 correspondences
Estimating Fundamental Matrix: The Eight-Point Algorithm
(Longuet-Higgins, 1981)

\[
(u, v, 1) \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = 0
\]

Minimize:
under the constraint

\[
(uu', uv', u, vu', vv', v, u', v', 1) \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0
\]

\[
\sum_{i=1}^{n} (p_i^T \mathcal{F} p_i')^2 = 1
\]
under the constraint

\[
|\mathcal{F}|^2 = 1.
\]
The 8-point Algorithm

8 corresponding points, 8 equations.

\[
\begin{pmatrix}
    u_1 u'_1 & u_1 v'_1 & u_1 & v_1 u'_1 & v_1 v'_1 & v_1 & u'_1 & v'_1 \\
    u_2 u'_2 & u_2 v'_2 & u_2 & v_2 u'_2 & v_2 v'_2 & v_2 & u'_2 & v'_2 \\
    u_3 u'_3 & u_3 v'_3 & u_3 & v_3 u'_3 & v_3 v'_3 & v_3 & u'_3 & v'_3 \\
    u_4 u'_4 & u_4 v'_4 & u_4 & v_4 u'_4 & v_4 v'_4 & v_4 & u'_4 & v'_4 \\
    u_5 u'_5 & u_5 v'_5 & u_5 & v_5 u'_5 & v_5 v'_5 & v_5 & u'_5 & v'_5 \\
    u_6 u'_6 & u_6 v'_6 & u_6 & v_6 u'_6 & v_6 v'_6 & v_6 & u'_6 & v'_6 \\
    u_7 u'_7 & u_7 v'_7 & u_7 & v_7 u'_7 & v_7 v'_7 & v_7 & u'_7 & v'_7 \\
    u_8 u'_8 & u_8 v'_8 & u_8 & v_8 u'_8 & v_8 v'_8 & v_8 & u'_8 & v'_8 \\
\end{pmatrix}
\begin{pmatrix}
    F_{11} \\
    F_{12} \\
    F_{13} \\
    F_{21} \\
    F_{22} \\
    F_{23} \\
    F_{31} \\
    F_{32} \\
\end{pmatrix}
= 
\begin{pmatrix}
    1 \\
    1 \\
    1 \\
    1 \\
    1 \\
    1 \\
    1 \\
    1 \\
\end{pmatrix}
\]

Invert and solve for \( \mathcal{F} \).

(Use more points if available; find least-squares solution to minimize \( \sum_{i=1}^{n} (p_i^T \mathcal{F} p'_i)^2 \) )
Properties of the Fundamental Matrix

- $F p'$ is the epipolar line associated with $p'$.
- $F^T p$ is the epipolar line associated with $p$.
- $F e' = 0$ and $F^T e = 0$.
- $F$ is singular.
Non-Linear Least-Squares Approach
(Luong et al., 1993)

Minimize

\[ \sum_{i=1}^{n} [d^2(p_i, Fp_i') + d^2(p_i', F^Tp_i)] \]

with respect to the coefficients of \( F \), using an appropriate rank-2 parameterization.
Problem with eight-point algorithm

Linear least-squares: unit norm vector $F$ yielding smallest residual

What happens when there is noise?
The Normalized Eight-Point Algorithm
(Hartley, 1995)

• Center the image data at the origin, and scale it so the mean squared distance between the origin and the data points is 2 pixels: \( q_i = T p_i \), \( q_i' = T' p_i' \).

• Use the eight-point algorithm to compute \( F \) from the points \( q_i \) and \( q_i' \).

• Enforce the rank-2 constraint.

• Output \( T^T F T' \).
The Normalized Eight-Point Algorithm
(Hartley, 1995)

Transform image to $\sim[-1,1] \times [-1,1]$

Least squares yields good results  
(Hartley, PAMI´97)
Epipolar geometry example

generic relations between two views is fully described by recovered 3x3 matrix $F$
Example: converging cameras
Example: motion parallel with image plane

If the camera translation is parallel to the $x$-axis, then $e' = (1, 0, 0)^T$, so (simple for stereo $\rightarrow$ rectification)

$$F = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}.$$
Example: forward motion
Fundamental matrix for pure translation

auto-epipolar
Fundamental matrix for pure translation

The epipole is fixed and is called Focus of Expansion
Visual cues (recall)

- Shading
- Texture
- Focus
- Motion
THE GEOMETRY OF MULTIPLE VIEWS

Motion Estimation

Lecture 11

Siamak Khatibi

BELEKINGE INSTITUTE OF TECHNOLOGY

Reading: Chapter 10.
Why estimate motion?

- Lots of uses
  - Motion Detection
  - Track object behavior
  - Correct for camera jitter (stabilization)
  - Align images (mosaics)
  - 3D shape reconstruction
  - Video Compression
Optical flow
Measurement of motion at every pixel
Optical flow

An image from Hamburg Taxi Sequence
Video Mosaics
Video Mosaics
Video Mosaics
Video Compression
Geo Registration

Results superimposed with the reference image
Video Segmentation
Structure From Motion
Optical flow
Measurement of motion at every pixel
Problem definition: optical flow

- How to estimate pixel motion from image $H$ to image $I$?
  - Solve pixel correspondence problem
    - given a pixel in $H$, look for nearby pixels of the same color in $I$

- Key assumptions
  - **color constancy**: a point in $H$ looks the same in $I$
    - For grayscale images, this is **brightness constancy**
  - **small motion**: points do not move very far

This is called the **optical flow** problem
Optical flow constraints (grayscale images)

Let’s look at these constraints more closely

- brightness constancy: Q: what’s the equation?

- small motion: (u and v are less than 1 pixel)
  - suppose we take the Taylor series expansion of I:

\[
I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \text{higher order terms}
\]

\[
\approx I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v
\]
Optical flow equation

- Combining these two equations

\[
0 = I(x + u, y + v) - H(x, y)
\]

shorthand: \( I_x = \frac{\partial I}{\partial x} \)

\[
\approx I(x, y) + I_x u + I_y v - H(x, y)
\]

\[
\approx (I(x, y) - H(x, y)) + I_x u + I_y v
\]

\[
\approx I_t + I_x u + I_y v
\]

\[
\approx I_t + \nabla I \cdot [u \ v]
\]

- In the limit as \( u \) and \( v \) go to zero, this becomes exact

\[
0 = I_t + \nabla I \cdot \left[ \frac{\partial x}{\partial t} \quad \frac{\partial y}{\partial t} \right]
\]
Optical flow equation

\[ 0 = I_t + \nabla I \cdot [u \ v] \]

- Q: how many unknowns and equations per pixel?
- Intuitively, what does this constraint mean?
  - The component of the flow in the gradient direction is determined
  - The component of the flow parallel to an edge is unknown
Aperture problem
Aperture problem
Solving the aperture problem

- How to get more equations for a pixel?
  - Basic idea: impose additional constraints
    - most common is to assume that the flow field is smooth locally
    - one method: pretend the pixel’s neighbors have the same \((u,v)\)
      - If we use a 5x5 window, that gives us 25 equations per pixel!

\[
0 = I_t(p_i) + \nabla I(p_i) \cdot [u \ v]
\]

\[
\begin{bmatrix}
I_x(p_1) & I_y(p_1) \\
I_x(p_2) & I_y(p_2) \\
\vdots & \vdots \\
I_x(p_{25}) & I_y(p_{25})
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
= -
\begin{bmatrix}
I_t(p_1) \\
I_t(p_2) \\
\vdots \\
I_t(p_{25})
\end{bmatrix}
\]

\[
A_{25 \times 2}
\quad
d_{2 \times 1}
\quad
b_{25 \times 1}
\]
How to get more equations for a pixel?

Basic idea: impose additional constraints
- most common is to assume that the flow field is smooth locally
- one method: pretend the pixel’s neighbors have the same \((u,v)\)

If we use a 5x5 window, that gives us 25*3 equations per pixel!

\[
0 = I_t(p_i)[0, 1, 2] + \nabla I(p_i)[0, 1, 2] \cdot [u \ v]
\]

\[
\begin{bmatrix}
I_x(p_1)[0] & I_y(p_1)[0] \\
I_x(p_1)[1] & I_y(p_1)[1] \\
I_x(p_1)[2] & I_y(p_1)[2] \\
\vdots & \vdots \\
I_x(p_{25})[0] & I_y(p_{25})[0] \\
I_x(p_{25})[1] & I_y(p_{25})[1] \\
I_x(p_{25})[2] & I_y(p_{25})[2]
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix} = -
\begin{bmatrix}
I_t(p_1)[0] \\
I_t(p_1)[1] \\
I_t(p_1)[2] \\
\vdots \\
I_t(p_{25})[0] \\
I_t(p_{25})[1] \\
I_t(p_{25})[2]
\end{bmatrix}
\]

\[
A \quad 75\times2 \\
d \quad 2\times1 \\
b \quad 75\times1
\]
Lukas-Kanade flow

- Prob: we have more equations than unknowns
  \[
  \begin{align*}
  \begin{bmatrix} A & d \end{bmatrix} & = b \\
  25 \times 2 & 2 \times 1 \\
  25 \times 1 & 
  \end{align*}
  \rightarrow \quad \text{minimize} \quad \|Ad - b\|^2
  \]

- Solution: solve least squares problem
  - minimum least squares solution given by solution (in d) of:
    \[
    \begin{bmatrix}
    \sum I_x I_x & \sum I_x I_y \\
    \sum I_x I_y & \sum I_y I_y
    \end{bmatrix}
    \begin{bmatrix}
    u \\
    v
    \end{bmatrix}
    = - \begin{bmatrix}
    \sum I_x I_t \\
    \sum I_y I_t
    \end{bmatrix}
    \]

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lukas & Kanade (1981)
Conditions for solvability

\[
\begin{bmatrix}
\sum I_x I_x & \sum I_x I_y \\
\sum I_x I_y & \sum I_y I_y
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
= -
\begin{bmatrix}
\sum I_x I_t \\
\sum I_y I_t
\end{bmatrix}
\text{equation}
\]

\[A^T A\]

\[A^T b\]

When is This Solvable?

- \(A^T A\) should be invertible
- \(A^T A\) should not be too small due to noise
  - eigenvalues \(\lambda_1\) and \(\lambda_2\) of \(A^T A\) should not be too small
- \(A^T A\) should be well-conditioned
  - \(\lambda_1/\lambda_2\) should not be too large (\(\lambda_1 =\) larger eigenvalue)
Eigenvectors of $A^T A$

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \ I_y] = \sum \nabla I (\nabla I)^T$$

- Suppose $(x,y)$ is on an edge. What is $A^T A$?
  - gradients along edge all point the same direction
  - gradients away from edge have small magnitude
    $$\left( \sum \nabla I (\nabla I)^T \right) \approx k \nabla I \nabla I^T$$
    $$\left( \sum \nabla I (\nabla I)^T \right) \nabla I = k ||\nabla I|| \nabla I$$
  - $\nabla I$ is an eigenvector with eigenvalue $k ||\nabla I||$
  - What’s the other eigenvector of $A^T A$?
    - let $N$ be perpendicular to $\nabla I$
      $$\left( \sum \nabla I (\nabla I)^T \right) N = 0$$
    - $N$ is the second eigenvector with eigenvalue 0
  - The eigenvectors of $A^T A$ relate to edge direction and magnitude
Edge

\[ \sum \nabla I (\nabla I)^T \]

- large gradients, all the same
- large \( \lambda_1 \), small \( \lambda_2 \)
Low texture region

\[ \sum \nabla I (\nabla I)^T \]
- gradients have small magnitude
- small \( \lambda_1 \), small \( \lambda_2 \)
High textured region

\[ \sum \nabla I (\nabla I)^T \]

- gradients are different, large magnitudes
- large \( \lambda_1 \), large \( \lambda_2 \)
Observation

- This is a two image problem BUT
  - Can measure sensitivity by just looking at one of the images!
  - This tells us which pixels are easy to track, which are hard
    - very useful later on when we do feature tracking...
Errors in Lukas-Kanade

- What are the potential causes of errors in this procedure?
  - Suppose $A^TA$ is easily invertible
  - Suppose there is not much noise in the image

- When our assumptions are violated
  - Brightness constancy is **not** satisfied
  - The motion is **not** small
  - A point does **not** move like its neighbors
    - window size is too large
    - what is the ideal window size?
Improving accuracy

- Recall our small motion assumption
  \[ 0 = I(x + u, y + v) - H(x, y) \]
  \[ \approx I(x, y) + I_x u + I_y v - H(x, y) \]
- This is not exact
  - To do better, we need to add higher order terms back in:
    \[ = I(x, y) + I_x u + I_y v + \text{higher order terms} - H(x, y) \]
- This is a polynomial root finding problem
  - Can solve using **Newton’s method**
    - Also known as **Newton-Raphson** method
  - Lukas-Kanade method does one iteration of Newton’s method
    - Better results are obtained via more iterations
**Iterative Refinement**

- Iterative Lukas-Kanade Algorithm
  1. Estimate velocity at each pixel by solving Lucas-Kanade equations
  2. Warp H towards I using the estimated flow field
     - *use image warping techniques*
  3. Repeat until convergence
Revisiting the small motion assumption

- Is this motion small enough?
  - Probably not—it’s much larger than one pixel ($2^{\text{nd}}$ order terms dominate)
  - How might we solve this problem?
Reduce the resolution!
Coarse-to-fine optical flow estimation

Gaussian pyramid of image H

Gaussian pyramid of image I

$u=10$ pixels

$u=5$ pixels

$u=2.5$ pixels

$u=1.25$ pixels

image H

image I
Coarse-to-fine optical flow estimation

Gaussian pyramid of image H

run iterative L-K

warp & upsample

run iterative L-K

Gaussian pyramid of image I

image H

image I
Multi-resolution Lucas Kanade Algorithm

Compute Iterative LK at highest level
For Each Level $i$
  • Take flow $u(i-1), v(i-1)$ from level $i-1$
  • Upsample the flow to create $u^*(i), v^*(i)$ matrices of twice resolution for level $i$.
  • Multiply $u^*(i), v^*(i)$ by 2
  • Compute $I_t$ from a block displaced by $u^*(i), v^*(i)$
  • Apply LK to get $u'(i), v'(i)$ (the correction in flow)
  • Add corrections $u'(i), v'(i)$ to obtain the flow $u(i), v(i)$ at $i^{th}$ level, i.e., $u(i) = u^*(i) + u'(i), v(i) = v^*(i) + v'(i)$
Optical Flow Results

Lucas-Kanade without pyramids
Fails in areas of large motion
Optical Flow Results

Lucas-Kanade with Pyramids
Multi-resolution Lucas Kanade Algorithm

Compute Iterative LK at highest level
For Each Level $i$

- Take flow $u(i-1), v(i-1)$ from level $i-1$
- Upsample the flow to create $u^*(i), v^*(i)$ matrices of twice resolution for level $i$.
- Multiply $u^*(i), v^*(i)$ by 2
- Compute $I_t$ from a block displaced by $u^*(i), v^*(i)$
- Apply LK to get $u'(i), v'(i)$ (the correction in flow)
- Add corrections $u'(i), v'(i)$ to obtain the flow $u(i), v(i)$ at $i^{th}$ level, i.e.,

$$u(i) = u^*(i) + u'(i), \quad v(i) = v^*(i) + v'(i)$$
Optical Flow Results

Lucas-Kanade without pyramids

Fails in areas of large motion
Optical Flow Results

Lucas-Kanade with Pyramids
Global Flow

- Dominant Motion in the image
  - Motion of all points in the scene
  - Motion of most of the points in the scene
  - A Component of motion of all points in the scene

- Global Motion is caused by
  - Motion of sensor (Ego Motion)
  - Motion of a rigid scene

- Estimation of Global Motion can be used to
  - Video Mosaics
  - Image Alignment (Registration)
  - Removing Camera Jitter
  - Tracking (By neglecting camera motion)
  - Video Segmentation etc.
Global Flow

Application: Image Alignment
Global Flow

- Special Case of General Optical Flow Problem
- Can be solved by using Lucas Kanade algorithm.
- Specialized algorithms exist that perform better by further constraining the problem.
Motion Models

Global Flow occurs because of 3D rigid motion of either the sensor or the scene.

- First we look for a parametric form of global flow vector.

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} = \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + T = R^T Z R^T Y R^T X \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + T
\]

3D Rigid Motion

\[
\begin{bmatrix}
\cos \beta \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma \\
\cos \beta \sin \gamma & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma \\
-\sin \beta & \sin \alpha \cos \beta & \cos \alpha \cos \beta
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} \approx \begin{bmatrix}
1 & -\gamma & \beta \\
\gamma & 1 & -\alpha \\
-\beta & \alpha & 1
\end{bmatrix} \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + \begin{bmatrix}
T_X \\
T_Y \\
T_Z
\end{bmatrix}
\]

\(\cos \theta \approx 1\) (If \(\theta\) is small)
\(\sin \theta \approx \theta\)
Also neglecting the higher order terms
Motion Models

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix}
\approx
\begin{bmatrix}
1 & -\gamma & \beta \\
\gamma & 1 & -\alpha \\
-\beta & \alpha & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
+ \begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
\]

3D Rigid Motion

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix}
\approx
\begin{bmatrix}
0 & -\gamma & \beta \\
\gamma & 0 & -\alpha \\
-\beta & \alpha & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
+ \begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
\]

Velocity Vector

\[
\begin{bmatrix}
V_x \\
V_y \\
V_z
\end{bmatrix}
= \text{Velocity Vector}
\]

Translational Component of Velocity

\[
\begin{bmatrix}
V_{T_x} \\
V_{T_y} \\
V_{T_z}
\end{bmatrix}
= \text{Translational Component of Velocity}
\]

Angular Velocity

\[
\begin{bmatrix}
\omega_x \\
\omega_y \\
\omega_z
\end{bmatrix}
= \text{Angular Velocity}
\]
Motion Models

Orthographic Projection

\[
\begin{bmatrix}
V_X \\
V_Y \\
V_Z
\end{bmatrix} \approx \begin{bmatrix}
0 & -\omega_Z & \omega_Y \\
\omega_Z & 0 & -\omega_X \\
-\omega_Y & \omega_X & 0
\end{bmatrix} \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + \begin{bmatrix}
V_{T_x} \\
V_{T_y} \\
V_{T_z}
\end{bmatrix}
\]

\[V_X = -\omega_Z Y + \omega_Y Z + V_{T_x}\]
\[V_Y = \omega_Z X - \omega_X Z + V_{T_y}\]
\[V_Z = -\omega_Y X + \omega_X Y + V_{T_z}\]

\[u = v_x = -\omega_Z y + \omega_Y Z + V_{T_x}\]
\[v = v_y = \omega_Z x - \omega_X Z + V_{T_y}\]
\[x = X\] (Orthographic Projection)
\[y = Y\]
Motion Models

Perspective Projection (Arbitrary Flow)

\[
x = f \frac{X}{Z} \\
y = f \frac{Y}{Z} \\
\]

\[
u = v_x = f \frac{ZV_x - XV_z}{Z^2} = f \frac{V_x}{Z} - \left( f \frac{X}{Z} \right) \frac{V_z}{Z} = f \frac{V_x}{Z} - x \frac{V_z}{Z} \\
v = v_y = f \frac{ZV_y - YV_z}{Z^2} = f \frac{V_y}{Z} - \left( f \frac{Y}{Z} \right) \frac{V_z}{Z} = f \frac{V_y}{Z} - y \frac{V_z}{Z} \\
\]

\[
V_x = -\omega_z Y + \omega_y Z + V_{Tx} \\
V_y = \omega_z X - \omega_x Z + V_{Ty} \\
V_z = -\omega_y X + \omega_x Y + V_{Tz} \\
\]

\[
u = v_x = \frac{V_{Tz} x - V_{Tx} f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x x y}{f} - \frac{\omega_y x^2}{f} \\
v = v_y = \frac{V_{Tz} y - V_{Ty} f}{Z} + \omega_x f - \omega_z x - \frac{\omega_y x y}{f} + \frac{\omega_x y^2}{f} \\
\]

Basic Equations of the Motion Field
Motion Models

Planar Scene + Orthographic Projection (Affine Flow)

\[
\begin{align*}
  u &= v_x = -\omega_z y + \omega_y Z + V_{T_x} \\
  v &= v_y = \omega_z x - \omega_x Z + V_{T_y}
\end{align*}
\]

\[
\begin{align*}
  u &= v_x = -\omega_z y + \omega_y (a + bx + cy) + V_{T_x} \\
  v &= v_y = \omega_z x - \omega_x (a + bx + cy) + V_{T_y}
\end{align*}
\]

\[
\begin{align*}
  u &= v_x = a_1 x + a_2 y + b_1 \\
  v &= v_y = a_3 x + a_4 y + b_2 \\
  \begin{bmatrix} u \\ v \end{bmatrix} &= A \begin{bmatrix} x \\ y \end{bmatrix} + B
\end{align*}
\]

\[
\begin{align*}
  Z &= a + bX + cY \quad \text{(Equation of Plane)} \\
  x &= X \\
  y &= Y \\
  a_1 &= b \omega_y \\
  a_2 &= c \omega_y - \omega_z \\
  b_1 &= a \omega_y + V_{T_x} \\
  a_3 &= \omega_z - b \omega_x \\
  a_2 &= -c \omega_x \\
  b_1 &= -a \omega_x + V_{T_y}
\end{align*}
\]
Motion Models

Planar Scene + Perspective Projection (Pseudo-Perspective)

\[ u = v_x = \frac{V_{Tz} x - V_{Tx} f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f} \]  
(Perspective Flow)

\[ v = v_y = \frac{V_{Tz} y - V_{Ty} f}{Z} + \omega_x f - \omega_z x - \frac{\omega_y xy}{f} + \frac{\omega_x y^2}{f} \]

\[ \frac{1}{Z} = \frac{1}{a} - \frac{b}{af} x - \frac{c}{af} y \]

\[ Z = a + bX + cY \]  
(Equation of Plane)

\[ x = f \frac{X}{Z}, \quad y = f \frac{Y}{Z} \]

\[ u = v_x = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy \]

\[ v = v_y = a_6 + a_7 x + a_8 y + a_4 xy + a_5 y^2 \]
Estimation of Global Flow

Assume Affine Flow:

\[
\begin{bmatrix}
    u \\
    v
\end{bmatrix} = A \begin{bmatrix}
    x \\
    y
\end{bmatrix} + B
\]

\[
u = v_x = a_1 x + a_2 y + b_1
\]

\[
v = v_y = a_3 x + a_4 y + b_2
\]

\[
\begin{bmatrix}
    u \\
    v
\end{bmatrix} = \begin{bmatrix}
    x & y & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & x & y & 1
\end{bmatrix} \begin{bmatrix}
    a_1 \\
    a_2 \\
    b_1 \\
    a_3 \\
    a_4 \\
    b_2
\end{bmatrix}
\]

\[
U = Xa
\]

Estimation of Global Flow

\[(\nabla I)^T \begin{bmatrix} u \\ v \end{bmatrix} + I_t = 0 \quad \text{(Optical Flow Constraint Equation)}\]

\[(\nabla I)^T \mathbf{X} \mathbf{a} + I_t = 0\]

\[
\sum_{(x,y) \in I} \left[ (\nabla I)^T \mathbf{X} \mathbf{a} + I_t \right]^2 = 0
\]

\[
\min \sum_{(x,y) \in I} \left[ (\nabla I)^T \mathbf{X} \mathbf{a} + I_t \right]^2 \quad \text{subject to} \quad \sum_{(x,y) \in I} \left[ (\nabla I)^T \mathbf{X} \mathbf{a} + I_t \right]^2 = 0
\]

\[
\begin{bmatrix}
\sum_{(x,y) \in I} \mathbf{X}^T (\nabla I)(\nabla I)^T \mathbf{X}
\end{bmatrix} \mathbf{a} = - \sum_{(x,y) \in I} I_t \mathbf{X}^T \nabla I
\]

\[
\mathbf{Aa} = \mathbf{B}
\]
Iterative Refinement

Iterative Algorithm

1. Estimate global flow by solving linear system $Aa=B$
2. Warp $H$ towards $I$ using the estimated flow
   - *use image warping techniques*
   - Repeat until convergence or a fixed number of iterations
**Basic Components**

Recall:

\[
Xa = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ b_1 \\ a_3 \\ a_4 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 = b \omega_y \\ a_2 = c \omega_y - \omega_z \\ b_1 = a \omega_y + V_{T_x} \\ a_3 = \omega_z - b \omega_x \\ a_2 = -c \omega_x \\ b_1 = -a \omega_x + V_{T_y} \end{bmatrix}
\]

\[
\sum_{(x,y) \in I} X^T (\nabla I) (\nabla I)^T X a = - \sum_{(x,y) \in I} I_t X^T \nabla I
\]

\[
Aa = B
\]

**Motion Estimation (Global Flow)**
Basic Components

- Pyramid Construction
- Motion Estimation
- Image Warping
- Coarse to Fine Refinement
Coarse-to-fine global flow estimation

Gaussian pyramid of image $H$

$u=10$ pixels
$u=5$ pixels
$u=2.5$ pixels
$u=1.25$ pixels

Gaussian pyramid of image $I$
Coarse-to-fine global flow estimation

Gaussian pyramid of image H

Gaussian pyramid of image I

Compute Flow Iteratively

warp & upsample

Compute Flow Iteratively
Result of Global Motion Estimation

Image ‘t’

Image ‘t+1’

Affine Model

output

4 Pyramids Level
5 Iterations/Pyramid Level
Video Mosaic
Estimation of Global Flow

Single Iteration

Image ‘t’

Compute $\mathbf{A}$ and $\mathbf{B}$

Solve $\mathbf{Aa} = \mathbf{B}$

Warp by $\mathbf{a}$

Image ‘t+1’
Estimation of Global Flow

Iterative

Initial Estimate \( \mathbf{a} = [a_1, a_2, b_1, a_3, a_4, b_2]^T \)

Image ‘t’

Warp by \( \mathbf{a} \)

Compute \( \mathbf{A} \) and \( \mathbf{B} \)

Solve \( \mathbf{A} \delta \mathbf{a} = \mathbf{B} \)

Warp by \( \delta \mathbf{a} \)

Image ‘t+1’
Estimation of Global Flow

Parameters Update

Initial Estimate  \[ \mathbf{a} = [a_1, a_2, b_1, a_3, a_4, b_2]^T \]

Computed Parameters  \[ \delta \mathbf{a} = [\delta a_1, \delta a_2, \delta b_1, \delta a_3, \delta a_4, \delta b_2]^T \]

Update Equations

\[
\begin{align*}
a_1 &= a_1 \delta a_1 + a_3 \delta a_2 + a_1 + \delta a_1 \\
a_2 &= a_2 \delta a_1 + a_4 \delta a_2 + a_2 + \delta a_2 \\
b_1 &= b_1 \delta a_1 + b_2 \delta a_2 + b_1 + \delta b_1 \\
a_3 &= a_1 \delta a_3 + a_3 \delta a_4 + a_3 + \delta a_3 \\
a_4 &= a_2 \delta a_3 + a_4 \delta a_4 + a_4 + \delta a_4 \\
b_2 &= b_1 \delta a_3 + b_2 \delta a_4 + b_2 + \delta b_2
\end{align*}
\]
Estimation of Global Flow

Initial Estimate: \( \mathbf{a} = [a_1 \ a_2 \ b_1 \ a_3 \ a_4 \ b_2]^{T} \)

Iterative

Image \( 't' \)

Warp by \( \mathbf{a} \)

Compute \( \mathbf{A} \) and \( \mathbf{B} \)

Solve: \( \mathbf{A} \delta \mathbf{a} = \mathbf{B} \)

Update \( \mathbf{a} \)

Iterate

Image \( 't+1' \)