Automatic Image Alignment (direct)

with a lot of slides stolen from Steve Seitz and Rick Szeliski

15-463: Computational Photography
Alexei Efros, CMU, Fall 2005
Today

Go over Midterm
Go over Project #3
Start Automatic Alignment

Reading:
  • Szeliski, Sections 3 and 4
How do we align two images automatically?

Two broad approaches:

- Feature-based alignment
  - Find a few matching features in both images
  - Compute alignment

- Direct (pixel-based) alignment
  - Search for alignment where most pixels agree
Direct Alignment

The simplest approach is a brute force search (hw1)

- Need to define image matching function
  - SSD, Normalized Correlation, edge matching, etc.
- Search over all parameters within a reasonable range:

  e.g. for translation:
  
  ```
  for tx=x0:step:x1,
      for ty=y0:step:y1,
          compare image1(x,y) to image2(x+tx,y+ty)
      end;
  end;
  ```

Need to pick correct \(x_0\), \(x_1\) and \(\text{step}\)

- What happens if \(\text{step}\) is too large?
Direct Alignment (brute force)

What if we want to search for more complicated transformation, e.g. homography?

\[
\begin{bmatrix}
wx' \\
wy' \\
w
\end{bmatrix} = \begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

for \( a = a_0 : a_{\text{step}} : a_1 \),
for \( b = b_0 : b_{\text{step}} : b_1 \),
for \( c = c_0 : c_{\text{step}} : c_1 \),
for \( d = d_0 : d_{\text{step}} : d_1 \),
for \( e = e_0 : e_{\text{step}} : e_1 \),
for \( f = f_0 : f_{\text{step}} : f_1 \),
for \( g = g_0 : g_{\text{step}} : g_1 \),
for \( h = h_0 : h_{\text{step}} : h_1 \),

compare \( \text{image1} \) to \( H(\text{image2}) \)
end; end; end; end; end; end; end; end; end;
Problems with brute force

Not realistic

- Search in $O(N^8)$ is problematic
- Not clear how to set starting/stopping value and step

What can we do?

- Use pyramid search to limit starting/stopping/step values
- For special cases (rotational panoramas), can reduce search slightly to $O(N^4)$:
  - $H = K_1R_1R_2^{-1}K_2^{-1}$ (4 DOF: $f$ and rotation)

Alternative: gradient decent on the error function

- i.e. how do I tweak my current estimate to make the SSD error go down?
- Can do sub-pixel accuracy
- BIG assumption?
  - Images are already almost aligned (<2 pixels difference!)
  - Can improve with pyramid

- Same tool as in **motion estimation**
Motion estimation: Optical flow

Will start by estimating motion of each pixel separately
Then will consider motion of entire image
Why estimate motion?

Lots of uses

- Track object behavior
- Correct for camera jitter (stabilization)
- Align images (mosaics)
- 3D shape reconstruction
- Special effects
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: \((u \text{ and } v \text{ are less than } 1 \text{ pixel})\)
  - 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

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

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

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

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

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

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

\[ 0 = I_t + \nabla I \cdot [\frac{\partial x}{\partial t} \ \frac{\partial y}{\partial t}] \]
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

This explains the Barber Pole illusion

http://www.sandlotscience.com/Ambiguous/barberpole.htm
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!

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

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

\[
A_{25\times2}
\]

\[
d_{2\times1}
\]

\[
b_{25\times1}
\]
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!

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

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

\[
\begin{bmatrix}
   A \\
   d
\end{bmatrix}
\begin{bmatrix}
   75 \times 2 \\
   2 \times 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
   b
\end{bmatrix}
\begin{bmatrix}
   75 \times 1
\end{bmatrix}
\]
Lukas-Kanade flow

Prob: we have more equations than unknowns

\[
\begin{align*}
A_{25 \times 2} \cdot d_{2 \times 1} &= b_{25 \times 1} \\
\text{minimize } &\| Ad - b \|^2
\end{align*}
\]

Solution: solve least squares problem

- minimum least squares solution given by solution (in \( d \)) of:

\[
\left( A^T A \right)_{2 \times 2} \cdot d_{2 \times 1} = A^T b_{2 \times 1}
\]

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

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

- Optimal \((u, v)\) satisfies Lucas-Kanade equation
  \[
  \begin{bmatrix}
  \sum I_x I_x & \sum I_x I_y \\
  \sum I_x I_y & \sum I_y I_y
  \end{bmatrix}
  \begin{bmatrix}
  u \\
  v
  \end{bmatrix} = -
  \begin{bmatrix}
  \sum I_x I_t \\
  \sum I_y I_t
  \end{bmatrix}
  \]
  \(A^T A\)
  \(A^T b\)

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)

\(A^T A\) is solvable when there is no aperture problem

\[
A^T A = \begin{bmatrix}
  \sum I_x I_x & \sum I_x I_y \\
  \sum I_x I_y & \sum I_y I_y
  \end{bmatrix} = \sum \begin{bmatrix}
  I_x \\
  I_y
  \end{bmatrix} [I_x \ I_y] = \sum \nabla I(\nabla I)^T
\]
Local Patch Analysis
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 $A^T A$ 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?
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 (2nd 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=1.25 pixels

u=2.5 pixels

u=5 pixels

u=10 pixels
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
Beyond Translation

So far, our patch can only translate in (u,v)

What about other motion models?

• rotation, affine, perspective

Same thing but need to add an appropriate Jacobian (see Table 2 in Szeliski handout):

\[ A^T A = \sum_i J \nabla I (\nabla I)^T J^T \]

\[ A^T b = -\sum_i J^T I_t (\nabla I)^T \]
Image alignment

Goal: estimate single $(u,v)$ translation for entire image
  - Easier subcase: solvable by pyramid-based Lukas-Kanade
Lucas-Kanade for image alignment

Pros:
- All pixels get used in matching
- Can get sub-pixel accuracy (important for good mosaicing!)
- Relatively fast and simple

Cons:
- Prone to local minima
- Images need to be already well-aligned 😊

What if, instead, we extract important “features” from the image and just align these?
Feature-based alignment

1. Find a few important features (aka Interest Points)
2. Match them across two images
3. Compute image transformation as per Project #3

How do we choose good features?

- They must prominent in both images
- Easy to localize
- Think how you did that by hand in Project #3
- Corners!
Feature Detection
Feature Matching

How do we match the features between the images?

• Need a way to describe a region around each feature
  – e.g. image patch around each feature
• Use successful matches to estimate homography
  – Need to do something to get rid of outliers

Issues:

• What if the image patches for several interest points look similar?
  – Make patch size bigger
• What if the image patches for the same feature look different due to scale, rotation, etc.
  – Need an invariant descriptor
Invariant Feature Descriptors