Yesterday.
How do you solve for a homography?
How else do you solve for a homography?

Chaining homographies,

\[
H_1 a = b \\
H_2 b = c \\
H_2 H_1 a = c
\]

Today. Optic Flow. (Finding nearby correspondences). Probably the last 2d day.
(Tomorrow, meaning thursday, stereo vision).

But first, an aside…
The Hockney Falco Thesis – Optics and projections have been used since 1400!
Aside over.

Examples of Motion Fields I

(a) Motion field of a pilot looking straight ahead while approaching a fixed point on a landing strip. (b) Pilot is looking to the right in level flight.

Examples of Motion Fields II

(a) Translation perpendicular to a surface. (b) Rotation about an axis perpendicular to image plane. (c) Translation parallel to a surface at a constant distance. (d) Translation parallel to an obstacle in front of a more distant background.

Optical flow

Motion Analysis Problems

• Correspondence Problem
  – Track corresponding elements across frames

• Reconstruction Problem
  – Given a number of corresponding elements, and camera parameters, what can we say about the 3D motion and structure of the observed scene?

• Segmentation Problem
  – What are the regions of the image plane which correspond to different moving objects?

The aperture problem
Aperture Problem

(a) Line feature observed through a small aperture at time $t$.
(b) At time $t + \delta t$ the feature has moved to a new position. It is not possible to determine exactly where each point has moved. From local image measurements only the flow component perpendicular to the line feature can be computed.

Normal flow: Component of flow perpendicular to line feature.

Optic Flow Constraint Equation
- Optic flow is 2d vector on image $(u,v)$
- Intensity at pixel $x,y$ is constant.
- $I_x u + I_y v + I_t = 0$
- Defines line in velocity space
- Require additional constraint to define optic flow.

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

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

- The summations are over all pixels in the $K \times K$ window.

Constant flow
- Prob: we have more equations than unknowns
  - The summations are over all pixels in the $25 \times 25$ window
- Solution: solve least squares problem
  - minimum least squares solution given by solution (in d) of:

\[
A^T A d = A^T I
\]

- One or both e.v. are 0
  - no corner, just an edge
  - no corner, homogeneous region

Taking a closer look at $(A^T A)$

The matrix for corner detection:

\[
A^T A = \begin{bmatrix}
\sum E_x^2 & \sum E_x E_y \\
\sum E_x E_y & \sum E_y^2
\end{bmatrix}
\]

is singular (not invertible) when $\det(A^T A) = 0$

But $\det(A^T A) = \prod \lambda_i = 0 \rightarrow$ one or both e.v. are 0

One e.v. = $0 \rightarrow$ no corner, just an edge
Two e.v. = $0 \rightarrow$ no corner, homogeneous region

Aperture Problem!
**Edge**
- large gradients, all the same
- large $\lambda_1$, small $\lambda_2$

**Low texture region**
- gradients have small magnitude
- small $\lambda_1$, small $\lambda_2$

**High textured region**
- 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...

**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?

**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
Reduce the resolution!

Coarse-to-fine optical flow estimation

Additional Constraints

- Additional constraints are necessary to estimate optical flow, for example, constraints on size of derivatives, or parametric models of the velocity field.
- Horn and Schunck (1981): global smoothness term

Optical flow result

Calculus, Schmalculus

Let $\nabla A = (\partial_x, \partial_y)^T$ denote the gradient of $A$

$$\int \int (\nabla E \cdot u + E_y)^2 + \lambda (\nabla u)^2 + (\nabla v)^2 \, dx \, dy \rightarrow \min$$

A. (init) Solve for blockwise optic flow.

B. For each pixel, update optic flow to be similar to neighbors, and (mostly) fit the optic flow constraint equation.
When a camera moves, a 3D scene point projects to different places on the image. This motion is called the optic flow. In the presence of noise, all methods to compute optic flow from images gives a bias.

Illusions.

Hajime Ouchi, 1977
Spillman, 1993
Hine, Cook & Rogers, 1995, 97
Khang & Essock, 1997
Pless, Fermuller, Aloimonos, 1999

• When a camera moves, a 3D scene point projects to different places on the image. This motion is called the optic flow. In the presence of noise, all methods to compute optic flow from images gives a bias.

Least squares solution

• One patch gives a system:

\[
\begin{bmatrix}
I_{x_1} & I_{y_1} \\
I_{x_2} & I_{y_2} \\
\vdots & \vdots \\
I_{x_n} & I_{y_n}
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
= 
\begin{bmatrix}
I_{t_1} \\
I_{t_2} \\
\vdots \\
I_{t_n}
\end{bmatrix}
= 0
\]

\[
\vec{u}_0 = \left( I_x^T I_x \right)^{-1} I_x^T I_t
\]

\[
\vec{u}_0 = M^{-1} I_x^T I_t
\]

Bias

y = ax + b, solve for a, b using least squares. If only y is messed up, you're golden (or blue).

If the x coordinates of your input is messed up, you're hosed. Because the least squares is minimizing vertical distance between the points and the line.
Taylor expansion around zero noise

- Assuming Gaussian noise, and small high order terms:
  \[ E\left( \tilde{\mathbf{u}} \right) = \mathbf{u}_0 + K_i \left( M^{-1} \mathbf{u}_0 \right) \sigma_n^2 \]
- Asymptotically true for any symmetric distribution (Stewart, 97).
- The expected bias can be explained in terms of the eigenvalues of \( M \).

Intermediate Distributions

- Bias affects length and direction of optic flow.
- Bias is dependent on magnitude of noise, and orientation of gradient distribution relative to actual optic flow.

In the Ouchi Pattern

- The change in gradient distribution leads to different biases in the computed optic flow.

Fixing the bias?

- Correcting the bias amounts to learning the distribution of the noise.
- In the case of total least squares, need to find the ratio of the variance of the noise in the spatial and temporal derivatives.
- Fourier Methods to compute flow also locally combine one-dimensional flow measurements. Unbiased error in spatiotemporal filter response leads to bias in optic flow estimate.

Avoiding the bias?

- A global model of the scene motion allows information to be combined over larger regions.
- Simultaneous estimation of structure and motion avoids motion discontinuities, allows larger patches and gives smaller derivative errors.
Prof. Pless,

In Presentation 8 slide 10, I think that the matrix is incorrect. It is 0 0 0 -x -y 1 xy' yy', I think it should be 0 0 0 -x -y -1 xy' yy'. The 1 should be negative instead of positive. In the equation above it is a -f not a +f. When I change the sign, my program works a lot better.

I believe you.

If you have a matrix equation

Ax = B

And you have A & B, do you calculate X by

B / A

or

B * A (inverse)?

Or are they the same thing? And how do you do that operation in Matlab?

Careful! given: Ax = B.

X = inv(A) * B

Or x = A \ B

Is there a recommended method for using Matlab to make a mosaic once we've done the transformation? It seems like matlab has built-in functionality for everything, but nothing for joining two images together. Thanks.

Sheesh, you think with all the help matlab is giving you, you could write the loop to go through and do the bilinear interpolation yourself.

There is a command called "interp2" which may be of interest. It may take longer to figure this out that it does to write the loop yourself.