Course notes, CS664, 11/2/04

- Administrivia: PS1 due today!

- Quiz 3 on Thursday. Coverage: expansion move algorithm.

- You will need to select your paper to write a 1 page report on shortly, and also pick a final project. Deadline for choice of both is a week from today.

- Useful hint: if $(50, 75)$ is at disparity 5 in the left image, then it corresponds to $(50 - 5, 75)$ in the right image. I think this is the bug many people are facing.

- Also note that the ground truth is registered with respect to one image (I think it's the left one, but you can easily check).

Optical flow recap

- The whole method works by taking the Taylor expansion and dropping terms higher than first order. This is a common tech-

nique; whenever you see it you should realize that local *linearity* is being assumed.

- Motivation for OFCE in 1D: consider a curve that moves to the right. We want to find u, the velocity (assumed constant). So $f(x)$ becomes $g(x-u)$, negative because we move to the right.

- Simple case: curve is a line. Slope of line is $I_x = m$.

- Assume observed intensity goes down, change in intensity is $-I_t$ (we will define I_t to be positive). Then by definition of the slope,

$$m = -\frac{I_t}{u} = I_x.$$

  This is the OFCE in 1D.

- More general case, still in 1D: curve is locally linear.

- As a rule, moving along the derivative doesn't move you along the curve, and the approximation gets worse the more the curve wiggles and the farther you move.

- Consequence: optical flow methods don't really work on textured images for "big" motions (more than 1 pixel). Often questionable even for smooth images and small motions...

Aperture problem

- Again consider a black line moving, seen from a restricted view. We know the velocity perpendicular to the line, but not along the line.

Lucas-Kanade

- A nice simple idea is to use least squares fitting locally (Lucas-Kanade).

- Assume that there is a single (u, v) for a few nearby pixels (local translational motion).

- At each pixel i, j we have the OFCE:

$$I_x(x_i, y_i) \cdot u + I_y(x_i, y_i) \cdot v = -I_t(x_i, y_i)$$

- We can write this set of equations in matrix form as

$$\begin{bmatrix} I_x(x_1, y_1) & I_y(x_1, y_1) \\ I_x(x_2, y_2) & I_y(x_2, y_2) \\ \vdots & \vdots \\ I_x(x_n, y_n) & I_y(x_n, y_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_t(x_1, y_1) \\ -I_t(x_2, y_2) \\ \vdots \\ -I_t(x_n, y_n) \end{bmatrix}$$

$$S \cdot \begin{bmatrix} u \\ v \end{bmatrix} = -T$$

- Note that S, T are observations, u, v unknowns. Overconstrained problem, in general.

- Consider Ax as a combination of the columns of A weighted by the elements of x. What about $Ax = y$ when there is no combination of the columns of A that equals y?

- The columns of A span a space that doesn't include y. Let's try to get as near as possible in the obvious (least squares) sense, i.e. minimize the squared difference

$$||y - A\hat{x}||_2^2.$$

  If you want you can think of this as energy minimization, or optimization.

- Note: in least squares you assume that A is right, and you want to get close to the space it spans. There is an interesting variant (much harder to solve) where you allow A to have errors as well [TLS].

- It has a simple solution (Gauss), namely $A^tAx = A^ty$, $\hat{x} = (A^tA)^{-1}A^ty$. Note that A^tA is square! Sometimes $\hat{x}$ is called the *pseudoinverse* of A.

- This has a natural interpretation in terms of line fitting (note, again, that errors (aka *residuals*) are only measured vertically).

- In the of motion, using least squares assumes that the constraint lines are right. For motion the pseudoinverse is particularly simple because

$$S^tS = \begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{bmatrix}$$

$$S^tT = \begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix}$$

  Moreover, inverting a 2x2 matrix has a simple closed form you learned in High School.

- Matrix is nearly uninvertible when the spatial gradients are all close to each other (draw a picture with similar constraint lines). In this case the local patch doesn't help you. Obviously you won't get a decent answer if the image gradient points uniformly in one direction.

- Related fact: the matrix S^tS is also used in a popular corner detection algorithm (KLT). The other extreme is a corner. The more "corner-like" the place where you are, the better optical flow estimates you will get.

Robust methods

- Major problem: some pixels may be wrong (have different motion u, v)

- Lots of methods are linear, and linear assumptions break down in the presence of outliers. Example: consider the mean of a 1D sample, or the least squares fit to a line. What happens if one point is wrong?

- We've seen this issue before, with the choices of V.

- If we know which pixels were less reliable we can weight their errors less.

- For a diagonal weighting matrix W, our energy function or error measure is

$$||W(y - A\hat{x})||_2^2.$$

  Weighted least squares solution is pretty similar,

$$\hat{x} = (A^t W^2 A)^{-1} A^t W^2 y$$

  But how do we know W?

- Iteratively Reweighted Least Squares: start with $W = I$ and solve. Figure out which pixels have high error (i.e., violate the OFCE). Reduce their weights and continue.

Parametric models

- Another place where similar optimization methods are used is in global motion estimation with a model (this can be done locally as well).

- Standard model is affine:

$$u(x, y) = a_1 + a_2 x + a_3 y$$

$$v(x, y) = a_4 + a_5 x + a_6 y$$

An affine motion maps between two arbitrary triangles, and can be defined by its action on a triangle.

- Affine motions exactly model a 3D plane under orthographic projection, which is exactly what Birchfield and Tomasi used in the last graph cuts lecture.

- We can plug this into the OFCE (the old case was $a_2 = a_3 = a_5 = a_6 = 0$). At each pixel i, j we get:

  $I_x(x_i, y_i) \cdot (a_1 + a_2 x + a_3 y) + I_y(x_i, y_i) \cdot (a_4 + a_5 x + a_6 y) = -I_t(x_i, y_i)$

- Now our minimization problem must find 6 parameters rather than 2, but we can still use least squares (or IRLS if we want).