

Computer Science 664
Fall 2002

Assignment 2

Due Date: October 31

Material: Lectures on motion estimation.

In this problem set you will implement and experiment with simple parametric motion estimation. We will consider only the problem of estimating the best global translation u^* mapping pixels of image I to pixels of image J . As discussed in class, more complex parametric motions such as affine, projective, or “plane plus parallax” can produce quite useful estimates of image motion for a wide range of problems including insertion of artificial objects into video, construction of panoramic mosaics, and detection and removal of moving objects in video.

Recall that applying the optical flow constraint equation, one can use local estimates of image derivatives to compute the least squares estimate of a parametric motion such as the translation u^* (or more complex transformations). However, this method assumes that the magnitude of the motion is small.

For each image pixel i , let Δx_i and Δy_i denote the finite difference approximation to spatial derivatives of I , and let Δt_i denote the finite difference approximate time derivative (the difference between I and J for pixel i).

The least squares estimate for the translation $u^* = \min_u \|Du - t\|$ where

$$D = \begin{bmatrix} \Delta x_i & \Delta y_i \\ \vdots & \vdots \\ \Delta x_n & \Delta y_n \end{bmatrix}$$

is an array of the spatial derivatives at each pixel and $t = [t_1, \dots, t_i]^T$ is a column vector of the time derivative at each pixel.

This minimizing value u^* can be found by the method of normal equations, $u^* = (D^T D)^{-1} D^T t$. Note that this is simply a 2×2 system, where

$$D^T D = \begin{bmatrix} \sum_i \Delta x_i^2 & \sum_i \Delta x_i \Delta y_i \\ \sum_i \Delta x_i \Delta y_i & \sum_i \Delta y_i^2 \end{bmatrix}$$

and

$$D^T t = \begin{bmatrix} \sum_i -\Delta t_i \Delta x_i & \sum_i -\Delta t_i \Delta y_i \end{bmatrix}.$$

Moreover, $(D^T D)$, is a symmetric 2×2 matrix and so its inverse can be computed easily in closed form (there is no need for general matrix inverse code).

1. Implement the normal equations method of solving for the best least squares estimate u^* of the translation from I to J , using the estimates of local derivatives. This need only work for small motions (a pixel or two).

Hand in the code for this, together with a written description of any issues you needed to address. Also hand in the estimate that you compute for the “tree” and “scene” image pairs.

2. The estimates computed above are only valid if the translations are small. In this part, you should implement a 5-level Gaussian pyramid (i.e., the coarsest scale images should be scaled by 2^5 from the original images), and perform motion estimation using the least squares method applied to the Gaussian pyramid.

Recall that a Gaussian pyramid is obtained by smoothing the image with Gaussian (use $\sigma = 1$) and then subsampling every-other pixel to obtain an image half as large. This is repeated to obtain another image that is again half as large, and so on. (Note: You can use the Gaussian smoothing function in the image libraries or your own from Assignment 1.)

As discussed in class, to estimate motion using a Gaussian pyramid, first estimate the motion at the coarsest scale. Then transform the image J by applying the inverse motion to J , at one finer scale. To apply the motion at one finer scale you must multiply it by 2, as the finer scale image has twice the resolution. Now compute the motion estimate at the finer level between I and the transformed J . To keep track of the overall motion, you must add this to the transformation you used to warp J . Continue this at each level of the pyramid until you reach the original images. The final result is the overall motion estimate, which need not be local because it is the sum of local transformations at various scales.

Hand in the code for this, together with a written description of any issues you needed to address. Also hand in the estimates that you compute for the “tree” and “scene” image pairs.

3. When there are pixels that do not fit the overall motion, a least squares fit may be fairly inaccurate. As discussed in class, iterative least squares (IRLS) can be used to discard outliers and obtain a better estimate of the translation. In this case, the computation is $u^* = (D^T W^2 D)^{-1} D^T W^2 t$ where W is a diagonal weight matrix. The i -th weight along the diagonal, w_i is based on the error in the motion estimate at the i -th image pixel. You should define and explain an error measure for the motion at a given pixel (larger values for more error). Be clear in your writeup.

Given your local error measure at each pixel i , call it r_i , let $w_i = 1$ if $r_i < c$ and let $w_i = c/r_i$ otherwise, where the “tuning constant” c is 1.5 times the median of the r_i ’s.

Again the overall system is 2×2 , and $D^T W^2 D$ is just like $D^T D$ above, except each element in each summation is scaled by w_i and analogously for $D^T t$.

IRLS starts with an initial motion estimate, which you should compute using your normal least squares method from Part 1. Then, use that u^* to compute local error measures r_i and a resulting weight matrix. Compute a new value of u^* using the weighted computation, and again compute new errors and weights, iterating until the change in u^* is small.

You should apply the IRLS method to obtain a motion estimate at each level of the Gaussian pyramid from Part 2.

An image with the error values r_i can be a useful way of seeing where in the image things do not match the overall motion. Where this image is bright (large values) motions do not match. Your code should output such an image for the original image level of the Gaussian pyramid.

For this part hand in your writeup, code, the error images, and the overall translation estimates you obtain for the “tree” and “scene” image pairs.

4. Given the final motion estimates, one can create synthetic images that superimpose two (or more in the case of sequences) images with one another. In this part, write code that creates a synthetic image based on the average of the corresponding pixels of I and J given a translation. Note that your translation will generally not be integer-valued, so you should interpolate a value of J based on four neighboring pixels in that image and then compute the average of that with each pixel of I .

Again discuss any issues you encounter. For this part hand in your writeup, code and the composite images for the “trees” and “scene” pairs given the best motion estimates that you computed in Part 3.