Introduction

In lecture 8, we considered the problem of matching intensities in one image to the intensities in a second image. (We returned to this problem briefly in lecture 9 when we discussed scale space, but today we will not discuss scale explicitly.) To find the local translation from image I to image J of pixels centered at point $\mathbf{x}_0$, one finds the shift $\mathbf{h}$ of a local neighborhood such that the sum of squared differences was minimized. We also considered a more general motion model which included rotation, scaling, and shear. This was used for *tracking* feature points over multiple frames where the objects can deform significantly, for example, because of perspective changes.

Today we will look at a very different approach to tracking under deformations, which uses histograms. The motivation is that often when objects move, the basic transformation models that we saw in lecture 8 don't hold. For example, think of a person walking. The person's arms and legs will move back and forth relative to each other and change from frame to frame. Parts may also disappear and reappear e.g. as the person swings their arm and it passes behind their torso. We do not necessarily want to model these details. Instead we might want to track the person as a whole.

Using histograms for tracking throws away some of the detailed spatial information about where the pixels are, which correspond to different parts of the object. There are still some constraints, though: the objects are spatially localized in the image and most parts of the object remain visible from frame to frame, even though their spatial relationships might vary.

Intensity Histograms in a Region of Interest (ROI)

Here we have a sequence of images $I_0, I_1, \ldots$. We will refer to the 2D pixel positions in each image with variable $\mathbf{x}$. We will follow the tradition (which I personally am not so happy with) of many papers in the tracking literature and use the value $\mathbf{y}$ as a position in image i of the object we are trying to track.

For most of what we will do today, we only need to talk about two images. The first image has a point $\mathbf{y}_0$ marked which is the center of a *region of interest* (ROI). We are looking for the position of the corresponding point in the second image. We search over the variable $\mathbf{y}$ to find the best position $\mathbf{y}_1$. More generally, we would have an ROI centered at position $\mathbf{y}_n$ in image frame n and we would be searching for a shifted $\mathbf{y}_{n+1}$ in image frame $n+1$.

For simplicitly, suppose that the neighborhood defining an ROI is disk shaped and has some radius, although this assumption is not crucial – we could use a square neighborhood instead. In the ROI we can define a histogram of features. This could be a histogram of intensities, or RGB values, or filter outputs e.g. gradient vectors. Let's work with the RGB values today. However, rather than working with a histogram of 256^3 bins for all possible RGB values, we work with some quantized set of bins. We will use 8 bins for each RGB value which means we have $8^3 = 512$ bins for our histogram. We will use the notation $u = bin(I(\mathbf{x}))$ for the histogram bin containing the RGB value $I(\mathbf{x})$ of pixel $\mathbf{x}$.

We use the familiar function $\delta()$, but we use it in a slightly different way. We could define the histogram of a whole image as:

$$hist(u) = \sum_{\mathbf{x}} \delta(u - bin(I(\mathbf{x}))) \ .$$

This would give a function of u which counts the number of pixels in the whole image whose RGB

values fall in each bin u. For tracking, we instead want to consider the histogram of pixels $\mathbf{x}$ in a region of interest (ROI):

$$hist(u;\mathbf{y}) = \sum_{\mathbf{x}\in ROI(\mathbf{y})} \delta(u - bin(I(\mathbf{x}))).$$

Note that a pixel is either in the ROI or not in the ROI, so we are not downweighting points that are further from $\mathbf{y}$.

Brute force tracking with simple histogram comparison

Let $\mathbf{y}_n$ be the position of the center of the ROI in frame I_n. We wish to find a nearby location $\mathbf{y}_{n+1}$ in frame $n+1$ such that the region of interest centered at $\mathbf{y}_{n+1}$ has the most similar histogram. Let

$$hist_n(u;\mathbf{y}_n) = \sum_{\mathbf{x}\in ROI(\mathbf{y}_n)} \delta(u - bin(I(\mathbf{x}))).$$

be the histogram for the ROI at $\mathbf{y}_n$ in image I_n and let

$$hist_{n+1}(u;\mathbf{y}) = \sum_{\mathbf{x}\in ROI(\mathbf{y})} \delta(u - bin(I_{n+1}(\mathbf{x}))).$$

be the histogram for an ROI at any location $\mathbf{y}$ in image I_{n+1}. So we could compute some measure of the similiarity of these two histograms by choosing the $\mathbf{y}$ value that maximizes this similarity. For example, you could take

$$\sum_u |hist_{n+1}(u;\mathbf{y}) - hist_n(u;\mathbf{y}_n)|.$$

One limitation of this similarity measure is that it gives equal histogram weight to all points in each ROI. We would intuitively like give more weight to points near the center of the ROI ($\mathbf{y}_n$), similar to what was done in the image registration methods, namely Lucas-Kanade. We will show how to do this below.

A second limitation is that it is brute force: we need to check every value of $\mathbf{y}$ and compute $hist(u;\mathbf{y})$ for each one. This is reminiscent of the registration problem where a brute force approach would be to check each possible shift $(\Delta x, \Delta y)$ and chose the shift that minimizes the sum of squared differences. We saw that the Lucas-Kanade approach is different: one make an approximation about smoothness and then solves a linear algebra problem to to estimate an $(\Delta x, \Delta y)$ which reduces the sum of squared errors, and iterates. As I will mention later (but omit the details), one can set up an analogous approach for tracking using histograms.

Weighted histogram approach

To deal with the first limitation, we redefine a histogram so that it give more weight to pixels $\mathbf{x}$ that are close to the center of the ROI. Consider a non-negative weighting function $W(\mathbf{x})$ that has a maximum at $\mathbf{x} = 0$ and

$$\sum_{\mathbf{x}} W(\mathbf{x}) = 1.$$

(If you read more about this histogram-based tracking method, then you'll see that it is common to use the word "kernel" and symbol K for the weighting function, but we'll stick with more familiar W.) For our purposes, you can assume this is a Gaussian.

We can now define a *weighted histogram* for each position $\mathbf{y}$:

$$p(u;\mathbf{y}) = \sum_{\mathbf{x}} W(\mathbf{y}-\mathbf{x})\ \delta(u - bin(I(\mathbf{x})))$$

This is a function of bin u, for each image position $\mathbf{y}$. We will justify using the symbol p for "probability" below. Before we do, we make a few notes:

First, if $W(\mathbf{x})$ were itself a delta function $\delta(\mathbf{x})$, then $p(u;\ \mathbf{y})$ would be a delta function in the feature space, namely $p(u\ ;\ \mathbf{y})$ would be the same as $hist(u;\mathbf{y})$. It would have the value 1 for the bin $u = bin(I(\mathbf{y}))$, and 0 for all other bins u' where $u' \neq bin(I(\mathbf{y}))$.

Second, the summation above seems similar to a spatial convolution. However, be careful: we are not blurring image intensities here. Rather, we are spatially blurring the count values within each bin u of the histogram. We take the delta function (value 1) for the binned feature value $u = bin(I(\mathbf{x}))$ and, rather than keeping the value 1 at position $\mathbf{x}$, we distribute it over all neighboring pixels $\mathbf{y}$ with spatial weight $W(\mathbf{y}-\mathbf{x})$. Note that if two pixels $\mathbf{x}$ and $\mathbf{x}'$ are in the ROI of $\mathbf{y}$ and if they have intensities $I(\mathbf{x})$ and $I(\mathbf{x}')$ that map to the same histogram bin index u, then both will contribute to $p(u;\mathbf{y})$, namely we would get the sum of the two weighting functions within bin u.

Let's now justify the usage of the term "probability", i.e. the notation $p(u;\mathbf{y})$. For each pixel $\mathbf{x}$ in an image $I(\mathbf{x})$, it is obvious that

$$\sum_u hist(u;\mathbf{x}) = 1$$

since each pixel's RGB value falls in exactly one histogram bin. A more subtle property is that, for each position $\mathbf{y}$,

$$\sum_u p(u;\mathbf{y}) = 1.$$

To understand why, think of the delta functions at the bin $u = bin(I(\mathbf{x}))$ for each image position $\mathbf{x}$. Each of these delta functions contributes a value $W(\mathbf{x}-\mathbf{y})$ to $hist(u;\mathbf{y})$. These values must sum to 1 because $\sum_{\mathbf{x}} W(\mathbf{x}-\mathbf{y}) = 1$, namely we are assuming that $W()$ is a weighting function that sums to 1, and shifting the $W()$ so it is centered at $\mathbf{y}$ doesn't change this property.

Another way to think about it: for each position $\mathbf{y}$, the weighting function $W(\mathbf{x}-\mathbf{y})$ that is centered at $\mathbf{y}$ defines an ROI, namely the neighborhood over which the weighting function is non-negative. For each pixel $\mathbf{x}$ in the ROI, there is an RGB value $I(x)$ at that position and hence there is a bin u associated with $\mathbf{x}$. That pixel $\mathbf{x}$ contributes a value $W(\mathbf{x}-\mathbf{y})$ to $p(u;\mathbf{y})$. The sum of contributions from all pixels $\mathbf{x}$ must sum to 1 (the above equation), since the weights sum to 1.

Finally, since the $p(u;\mathbf{y})$ values are non-negative, as a function of u, $p(u;\mathbf{y})$ obeys the definition of a probability function – namely it is non-negative and sums to 1. So we can refer to it as such. This doesn't mean we should think of it as a probability function, like we would when flipping a coin or rolling dice. The theory is not trying to express this sort of randomization here. Rather we are trying to capture that the locations of points can change from one frame to another and we are modelling our uncertainty about the location of RGB values within a local region.

How to use these weighted histograms for tracking?

We are thinking of $p(u;\mathbf{y})$ for each $\mathbf{y}$ as *weighted histograms*, namely they describe the distribution of RGB binned values $u = bin(I(\mathbf{x}))$ for $\mathbf{x}$ in the neighborhood of $\mathbf{y}$, where we have weighted the binned values by their image distance from $\mathbf{y}$, namely weight $W(\mathbf{x}-\mathbf{y})$.

We consider these weighted histograms at different $\mathbf{y}$ values, similar to what we discussed in the brute force tracking earlier. We are given an image I_n and we have a center of an ROI at $\mathbf{y}_n$. We would like to find the position $\mathbf{y}$ in image I_{n+1} such that the weighted histogram

$$p_{n+1}(u;\mathbf{y}) = \sum_{\mathbf{x}} W(\mathbf{y}-\mathbf{x})\ \delta(u - bin(I_{n+1}(\mathbf{x})))$$

is most similar to the weighted histogram

$$p_n(u;\mathbf{y}_n) = \sum_{\mathbf{x}} W(\mathbf{y}_n-\mathbf{x})\ \delta(u - bin(I_n(\mathbf{x})))$$

at position $\mathbf{y}_n$ in image I_n, This position $\mathbf{y}$ will become $\mathbf{y}_{n+1}$.

Bhattacharya coefficient for similarity of probability functions

Given two probability functions $p(u)$ and $q(u)$ that are defined on u, the Bhattacharya coefficient is one way to define their similarity. It does so by taking the sum of the *geometric means* of the probabilities over all bins u, namely:

$$BC(p,q) = \sum_u \sqrt{p(u)\ q(u)}$$

For example, if p and q are the same probability functions, then the similarity will be 1, and if p and q are only non-zero at different values of u then the similarity will be 0. BC cannot be negative.

Tracking based on weighted-histogram similarity

Let $\mathbf{y}_n$ be the position of the center of the ROI in frame I_n. We wish to find a nearby ROI in frame $n+1$ which has the most similar weighted histogram.

Let $p_n(u;\mathbf{y}_n)$ be the weighted histogram in the ROI at position $\mathbf{y}_n$ in frame n, and let $p_{n+1}(u;\mathbf{y})$ be the weighted histogram at a general position $\mathbf{y}$ in frame $n+1$. The brute force approach would be to compute the Bhattacharya coefficient for all of the different $\mathbf{y}$ positions near $\mathbf{y}_n$,

$$BC(p_n(u;\mathbf{y}_n), p_{n+1}(u;\mathbf{y})) = \sum_u \sqrt{p_n(u;\mathbf{y}_n)\ p_{n+1}(u;\mathbf{y})}$$

and then choose the $\mathbf{y}$ that maximizes this similarity. This would be a relatively expensive approach.

One proposed method for doing this is to take a small step $\Delta\mathbf{y}$ in the direction of the *gradient* of BC. This is the basic idea of the strategy taken by the mean shift method described in a paper "Kernel-Based Object Tracking" by Comaniciu, Ramesh, and Meer in the early 2000's. The algorithm itself is relatively efficient and produces very good results, but the details are quite technical and would take a whole lecture to explain. If you would like to learn about it, see Mubarak Shah's video (start at 5:00 in)

