Using Mean Shift for Video Image Segmentation

Joel Darnauer
joeld@stanford.edu
650 714 7688

ABSTRACT

The goal of this project was to develop a fast video image segmentation routine which could be used as a pre-processing step for motion tracking. We chose mean shift [1] as the primary algorithm. Our implementation includes several enhancements including dynamically adjusting the kernel bandwidth based on the overall level of image noise, and keeping a cache of past moves to avoid repeated computations. The frame rate of the resulting implementation is still slow due to the quadratic complextity and iterative nature of the underlying algorithm, but provides a reasonable starting point for refinements. We conclude with a discussion of several limitations of the mean shift as a procedure for image segmentation and present a couple options for improvement.

GENERAL PROBLEM FORMULATION

Image segmentation is an important low-level task in computer vision. As we saw in the homework the k-means algorithm can be used to compress an image into a reduced color space that approximately represents the original image. A robust segmentation could be the first stage in a object tracking pipeline, and in some ways represents a dual approach to the more common approach of tracking corner features.

Informally, we define the video image segmentation problem as follows.

INPUT:

A set of *F* input frames containing *W* x *H* rgb pixels.

In the nomenclature of the class, the set of training examples X is *x_i*:{pos_x, pos_y, t, r, g, b}. Additional features could be computed (such as converting the color space from RGB to HSV or other transformations).

In our case we know that each frame covers pos_x and pos_y, which may lead to some interesting optimizations.

Due to time limitations in the project we were only able to process frames individually, so the "t" dimension will be ignored for now.

OUTPUT:
A labeling of each pixel in each frame to a segment-id. In a real-time algorithm the labeling of segments should be produced in a causal manner – that is – only the information in prior frames can be used to assign a label to a object.

Once this basic labeling is produced it should be possible to produce segmented video with diagnostic output showing motions of segments. Higher level algorithms could build upon this system to find intersections between segements an track multi-segment objects.

An alternative formalation of the segmentation problem presented in [3] is to consider image space as a 3-dimensional regular graph between neighboring pixels in x,y,t. The image segmentation problem consists of selecting edges in the graph which connect segments, or additionally of finding a spanning forest of the image graph. This interpretation will be important for some of our later optimizations.

Having considered the format of the output, we must now turn our attention to the properties of the labeling or spanning forest produced.

Intuitively we want pixels to be labeled in the same segment if they correspond to the same shaded object from frame to frame. While subjective pixels in the same segment should have the following features:

- They are spatially contiguous within the same frame.
- Should be of similar color, with possible gradients due to lighting (and optionally texture).
- Between frames there is a simple model of perspective-like transformations which explain the trajectory of the 3D surface for the segment, with allowances for deformation, occlusion and camera motion.
- A more complicated algorithm might include terms to compensate for global lighting fluctuations (flashing lights, camera exposure changes, etc).
- A *very* sophisticated algorithm might include processing to account for motion blur.

A number of these features are beyond the scope of this class. For this class we will focus initially on the first problem of segmenting a still image.

THE MEAN SHIFT PARADIGM

Comaniciu [1] use the mean shift algorithm for image segmentation. Unlike k-means which might use a straight distance metric to assign pixels to a pre-defined number of clusters, the mean shift associates each pixel or feature with some local maximum in the feature density of feature space. “Feature density” is computed by convolving each feature in X with a kernel function, typically multivariate gaussian. The mean shift procedure performs gradient ascent in this feature density space, so each feature is associated with the point which is the attractor for the local basin of attraction that the feature resides in.

But why is this a good choice? Consider an input image which consists of a gray patch with a uniform gradient in the x-dimension from light to dark. A human viewer would attribute this gradient to some natural lighting variation and would associate all of the pixels in the patch with the same object. However pixels on the far left and far right of the patch have relatively large differences in color and spatial feature dimensions, so a naïve distance function might classify these pixels to different features. We need a way to unify these pixels based on the information between them.

Mean shift accomplishes this because in the gradient example above, the features form a uniform patch in the 5-dimensional feature space. If the kernel function has a sufficiently large aperature the maximum of the density will be at the center of the patch. (Figure 1.)

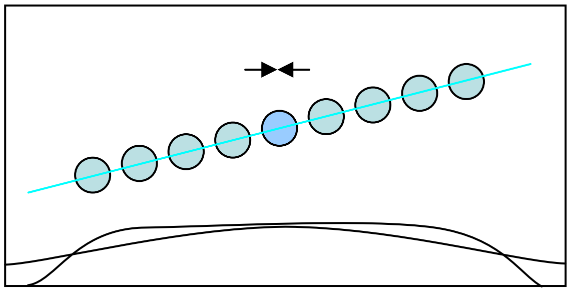

Figure 1. Blue dots are points in a simplified 2-d feature space where the vertical axis represents color and the horizontal axis represents space. In this case the points form a uniform gradient like the gray patch in the example. If the kernel function is wider than the patch, the center of the patch will be attractor. If the kernel function has a finite radius R then all the points which are further than r from the edge will have the same local density, and will therefore be stationary.

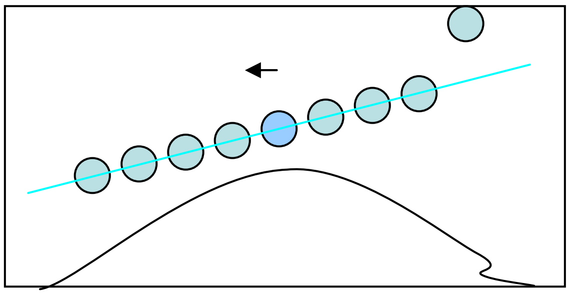

Figure 2: A discontinuity in the previous patch produces a dip in the density function which pushes points away from the edge. Because the features are uniformly dense in x-y, the mean shift operator can be thought of as a edge detector with a resolution determined by the spatial radius of the kernel.

So the basic mean shift operation is like edge detection and importantly is stationary for patches with a uniform first derivative.

The world view of mean shift in some ways is like mixture of gaussians, in that the modes of the feature density function are like the centers of the gaussian sources in MOG. The main difference is that the attractors in MOG can be very complex in shape. Later we will see that there are limitations to this view of the problem.

MEAN SHIFT PROCEDURE AND COMPLEXITY

The general mean shift procedure for a feature x_i procedes as follows:

1. For each feature in x_j in X, compute the distance d_j = x_j – x_i
2. Apply the kernel function k() to each d_j to compute the weight for that feature: w_j = k(d_j)
3. Compute a new x_i’ = $\sum_j$ w_j x_j / $\sum_j$ w_j
4. Repeat step 1 with the new x_i’ until |x_i-x_i’| is within some tolerance delta.

At each update the x_i’ is a weighted sum of the x_i. This procedure is guaranteed to converge as long as the kernel function k() is monotonically decreasing and convex. Typical kernels are guassian functions with some variance assigned to each dimension.

Two major difficulties with mean shift include the problem that the basic computation of the kernel distances are O(N^2) in the size of features, and that the number of iterations required is input dependent. The first advantage can be addressed by choosing a kernel that decays rapidly in certain dimensions. In practice the number of iterations for images is roughly constant and depends on the tolerance used, among other features.

If the kernel function used has a finite radius R, then the complexity is more like *O(N R^2 I)* where *I* is the mean number of iterations needed for convergence.

POST PROCESSING STEPS

Once the mean shift procedure is performed we have associated each pixel with an attractor. Especially in the case of small kernel width R, many points in flat patches will be stationary and should be associated with each other. A post processing step is done to merge all 4-connected clusters of attractors and then points are labeled with the ids of these superclusters. Further postprocessing operations like removing clusters below a certain size or minimum with could be performed at this point, but we avoided this to see the naked performance of the algorithm.

THE PIPELINE

The steps in the processing pipeline in our implementation can be seen in figure 3 and 4. This sequence uses a guassian kernel truncated at R=6 with some of the optimizations described below.

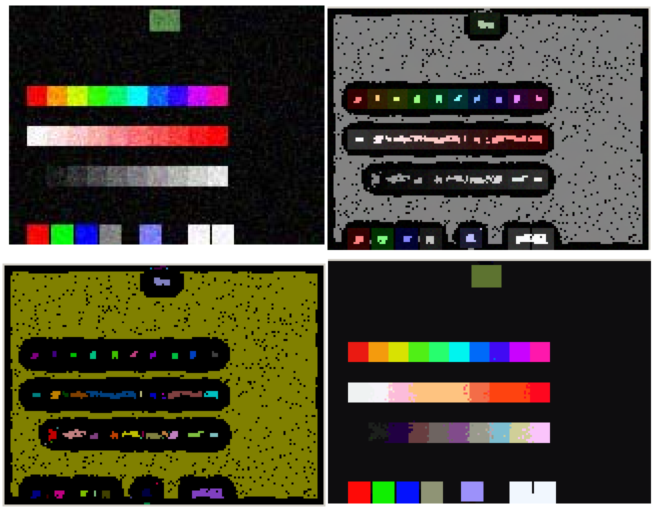

Figure 3. Processing pipeline for a test image sequence. Original test image (upper left). Attractors labeled with brightness in the color of the source image (upper right). Attractors labeled with the super-cluster after merging (lower left). Final segmented image (lower right). Some artifacts are visible in this segmentation due to limitations of the algorithm.

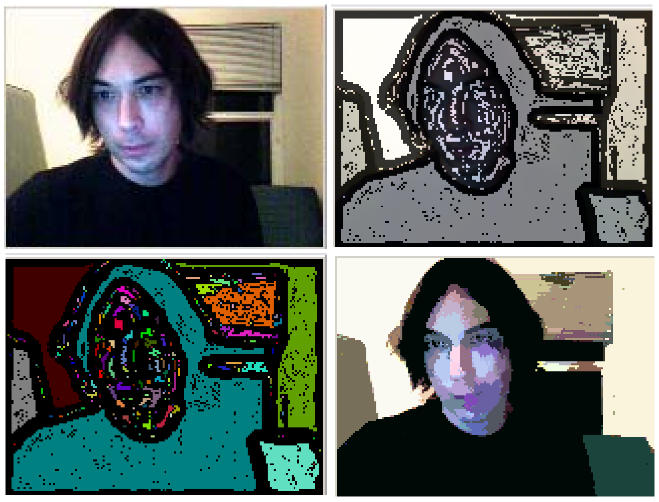

Figure 4. Same sequence of operations on a real webcam image.

NOISE OPTIMIZATION

One very important feature of the work was building a routine which could generate several abitrary test images. We included a feature that allowed us to add noise to the images and noticed that the vanilla gaussian kernel produced rather poor results because it assigned too much significance to these small fluctuations, especially in large flat spots.

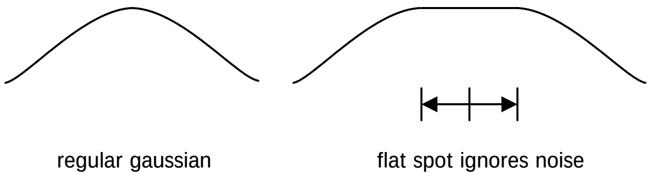

Figure 5. One improvement over the original mean shift algorithm was to discount small difference in color as being insignificant. The normal guassian kernel in color space is replaced with one with a flat spot whose width depends on the pixel-to-pixel variances thoughout the source image.

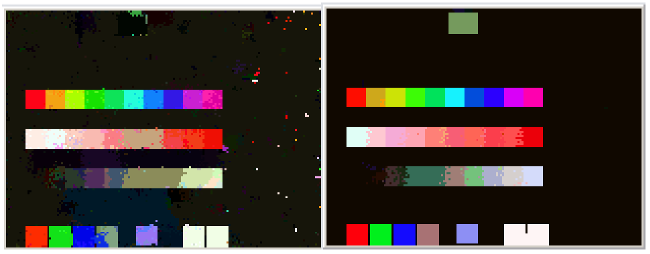

Figure 6. Result of the adaptive noise threshold. Left image is the standard guassian kernel. Right image is with the optimization.

We experimented with several kernels including faster ones such as 1/d, 1/1+d and k-d. All performed similarly, but we used the guassian kernel for most of our evaluation because it is more common and the performance difference were slight.

OPTIMIZING R

Since the inner loop depends quadratically on the kernel radius R, it is important to understand what value of R is necessary. Figure 7 (last page) shows a sweep of some images with different values of R. Generally a value of R=5 or R=6 produced reasonable results for our webcam images.

OPTIMIZING I

Since the mean shift computation moves features to a basin of attraction, we reasoned that many of these moves are likely to collect into streams and that it might be possible to speed up the computation by creating a move cache for the image. At each step of the mean shift, the cache is consulted and if the current feature position is within some distance d from one in the cache, we substitute the stored move. The cache is efficient because we store only one move for each xy position, reasoning that colors along a trajectory are likely to be similar.

There is a slight speedup from this approach because we are able to match about 30% of moves without noticable changes in image quality. The amount of additional storing and branching slows the algorithm down though, so the overall performance gain is only about 20%.

One interesting side effect of this is that the moves tend to be from neighboring pixel to neighboring pixel, so the result is to build up a graph of associations between pixels much like that in [3].

OVERALL PERFORMANCE

The frame rate of the overall performance is about 1100msec (0.9fps) on 120x160 pixel images for a kernel radius of 5 and about 8 iterations per pixel. The actual rate is image dependent, but corresponds to an inner loop time (computing the distance and kernel weights for each pixel) of about 90nsec. Since a call to exp() on my machine takes about 40nsec, this seems like reasonable performance. More work is needed to improve the inner loop time.

FUTURE DIRECTIONS

There are several interesting directions the project could take from here:

- There are a number of parameters in the algorithm which much be manually tweaked, which is quite unsatisfying. It would be better pehaps to use the test image procedure to create images with a known ground truth and then use supervised learning to create the images. If this is done, other algorithms such as SVN could be applied to the problem directly, which might produce much better results.
- Instead of trying label pixels or edges, the real problem is perhaps better formulated as finding the set of shaded polygons which are most likely to have generated the source image. I’m not sure what the parameter space of such an algorithm might look like, but it could also use the supervised learning approach, training on CG images from video games and movies.

If you have any interest or advice, please contact me at joeld@stanford.edu or call me at 650 714 7688.

REFERENCES

1. Dorin Comaniciu and Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. IEEE Trans Pattern Analysis and Machine Intelligence. 2002.
2. D. Comaniciu, V. Ramesh, and P. Meer, "*Kernel-based object tracking*," IEEE Trans. Pattern Anal. Machine Intell., vol. 25, pp. 564--577, 2003.
3. Pedro F. Felzenszwalb and Daniel P. Huttenlocher International Journal of Computer Vision, Volume 59, Number 2, September 2004

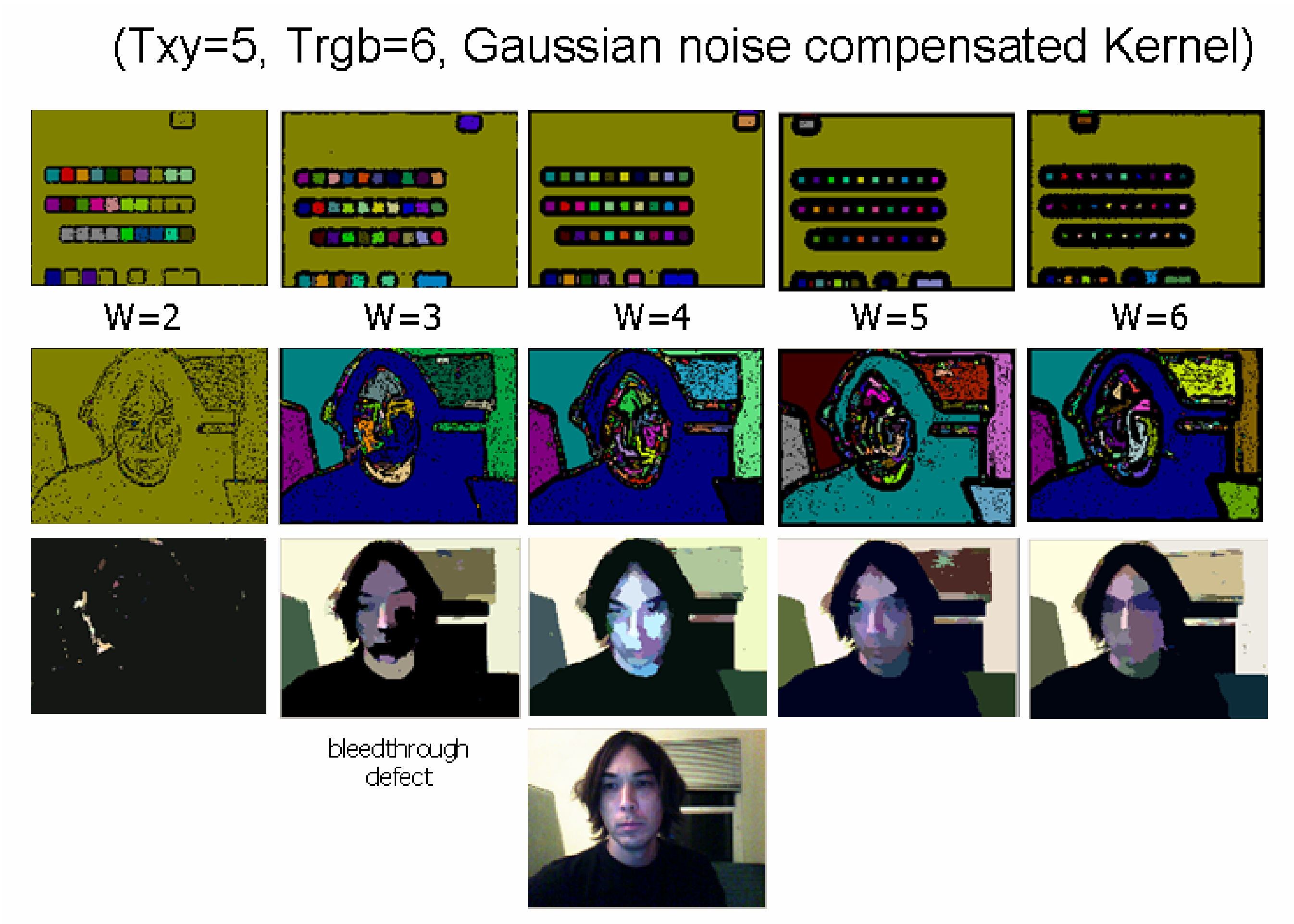

Figure 8: Selection of kernel radius R (here labeled W) and its effect on segmentation in test and real images. If W is too small, there is insufficient separation between clusters and bleedthrough defects are common.