Edge Detection using Mean Shift Smoothing

By Edan Lerner

1. Introduction

Edges characterize boundaries, which define the important structural properties of an image. Therefore edge detection is a problem of fundamental importance in image processing; many tasks in image processing, to be performed successfully, depend on a reliable edge detection mechanism. Edges in images are areas with strong intensity contrasts – a jump in intensity from one pixel to its neighbor. There are many ways to perform edge detection, most of which are based on calculating some sort of a numeric estimation of the intensity map gradient, and finding local maxima points, or zero crossing of the divergence of this gradient.

Probably the most difficult obstacle to overcome in the edge detection problem is the presence of noise in the image. This noise can often cause pixels that are not, by any means, edge pixels, to be detected as these. This undesirable phenomena is somewhat reduced by a process of smoothing of the image by some smoothing filter, such as a Gaussian filter. However, any smoothing filter will also reduce the real edges' contrast, resulting in a possible failure of detecting some of the real image edges. The delicate tradeoff between filtering out noise while preserving real edges' contrast is hard to control automatically, which is why it is handled many times by defining threshold parameters. The quality of the edge detection may heavily depend on these chosen parameters, which are hard to evaluate before any image processing has been preformed.

In the following sections, an algorithm for edge preserving smoothing is presented, based on the mean shift clustering algorithm. A short description of the edge detector, implemented as part of this project, is then given. Unlike the conventional smoothing filters, mean shift smoothing isn't performed as a convolution operation on the image map. Instead, data vector points are defined by the image map, rescaled, and shifted using the mean shift procedure (see section 2). The result of the procedure is a smoothed image, preserving a significant percentage of the real image edges, and occasionally even amplifying their contrast. In this case, as in many other cases, a parameter h that highly influences the smoothing result is required, though reasonable results are obtained by calculating h as a function of the image size.

2. Mean Shift Clustering

The clustering problem is defined as the following: given a data point set $\{\mathbf{x}_i\}$ in a d-dimensional Euclidean space R^d, assign a label l_i to each point $\mathbf{x}_i$, based on proximity to high density regions in the vector space. The number of different labels l_i depends on the characteristics of the data point set, as well as on the clustering algorithm.

The clustering problem is dealt with by numerous algorithms, of which many require different parameters to be inputted beforehand (such as the number of expected clusters). Mean shift clustering requires too a parameter h, known as the *window radius*. This parameter has a direct influence on the resolution of the cluster detection by the algorithm, as will be shown in the following sections.

We define the *multivariate kernel density estimate*, obtained with kernel $K(\mathbf{x})$ and window radius h, computed at the point $\mathbf{x}$, as:

$$f\left(\mathbf{x}\right)=\frac{1}{nh^{d}}\sum_{i=1}^{n}K\left(\frac{\mathbf{x}-\mathbf{x}_{i}}{h}\right) \tag{1}$$

For our discussion, we limit ourselves to the *Epanechnikov kernel*:

$$K_{E}\left(\mathbf{x}\right)=\begin{cases}\left(2c_{d}\right)^{-1}\left(d+2\right)\left(1-\mathbf{x}^{T}\mathbf{x}\right) & \text{if } \mathbf{x}^{T}\mathbf{x}<1\\ 0 & \text{otherwise}\end{cases} \tag{2}$$

where c_d is the volume of a unit d-dimensional sphere. Since the Epanechnikov kernel $K_E(\mathbf{x})$ is differentiable, we can derive the *mean shift vector* based on the gradient of $f(\mathbf{x})$:

$$\mathbf{M}_{h}\left(\mathbf{x}\right)=\frac{h^{2}}{d+2}\frac{\nabla f\left(\mathbf{x}\right)}{f\left(\mathbf{x}\right)}=\frac{1}{n_{\mathbf{x}}}\sum_{\mathbf{x}_{i}\in S_{h}\left(\mathbf{x}\right)}\mathbf{x}_{i}-\mathbf{x} \tag{3}$$

where the region $S_h(\mathbf{x})$ is a d-dimensional sphere of radius h, having the volume $h^d c_d$, centered at $\mathbf{x}$, and containing $n_{\mathbf{x}}$ data points (see [1] for detailed presentation). The mean shift vector has the direction of the density estimate's gradient at $\mathbf{x}$, when obtained with the Epanechnikov kernel. Since it always points towards the maximum increase of density, it can define a path leading to a local density maximum.

We now define the *mean shift procedure* as computation of the mean shift vector $\mathbf{M}_h(\mathbf{x})$, and translation of the window $S_h(\mathbf{x})$ by $\mathbf{M}_h(\mathbf{x})$. A series of successive iterations of the mean shift procedure is guaranteed to converge, as proved in [1]. Different variations of clustering algorithms are formulated based on this convergence; applying the procedure iteratively on data points will "displace" them to high data point density regions, depending on their location. Data points belonging to the same cluster will then converge to the same areas.

3. Image Smoothing by the Mean Shift Procedure

We will restrict the discussion to grey scale images, though generalizing the smoothing process for color images is trivial. Given an image map $I(i,j)$ (matrix of which each element represents its corresponding pixel's grey level intensity), a set of data points can be constructed by simply assigning each pixel's location in the map as the first two coordinates, and setting the third coordinate to be the normalized value of the pixel's intensity: for the (i,j)'th pixel of the image, the corresponding data point will be:

$$I(i,j) \rightarrow (i, j, I(i,j) * C) \tag{4}$$

where C is the normalization constant, chosen to be - [average of width and height of image]/[max intensity]:

$$C = \frac{(height + width)}{2} * \frac{1}{255} \tag{5}$$

The mean shift iteration (3) described above can be given explicitly:

$$\mathbf{y}_{k+1} = \frac{1}{n_k} \sum_{\mathbf{x}_i \in S_h(\mathbf{y}_k)} \mathbf{x}_i \tag{6}$$

In the $(k+1)$'th iteration, we shift the current location by the mean position of all data points contained within the sphere of radius h, centered at $\mathbf{y}_k$. The smoothing algorithm consists of the following steps:

For each $j = 1..n$

1. Initialize $k = 1$ and $\mathbf{y}_k = \mathbf{x}_j$.
2. Repeat: Compute $\mathbf{y}_{k+1}$ using the mean shift procedure (5); $k \leftarrow k+1$; until convergence.
3. Assign $I_{smoothed}(\mathbf{x}_j(1), \mathbf{x}_j(2)) = \mathbf{y}_k(3)$.

In the implementation of this algorithm, the default value for h was set to be half the square root of the average of the image's width and height:

$$h(height, width) = \frac{1}{2}\sqrt{\frac{width + height}{2}} \tag{7}$$

Other choices of functions could yield different results, but since running time is a crucial factor in this problem, keeping the value of h reasonably small is essential. Moreover, the greater the value of h is, the smaller the resolution of the resulting clusters, which may lead to deterioration of the edge detection, which is the motivation for the smoothing process. The following figures demonstrate the significant influence various h values have on the outcome of the smoothing process:

Original Image:

$$h = \frac{1}{4}\sqrt{\frac{height + width}{2}}$$

$$h = \frac{1}{2}\sqrt{\frac{height + width}{2}}$$

$$h = \sqrt{\frac{height + width}{2}}$$

Original Image

$$h = \frac{1}{4}\sqrt{\frac{height + width}{2}}$$

$$h = \frac{1}{2}\sqrt{\frac{height + width}{2}}$$

$$h = \sqrt{\frac{height + width}{2}}$$

Original Image:

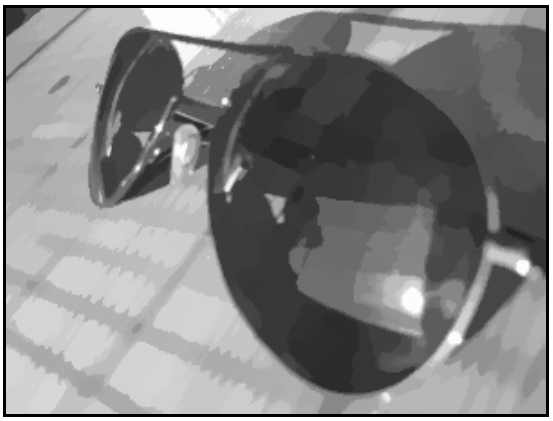
$$h = \frac{1}{4}\sqrt{\frac{height + width}{2}}$$

$$h = \frac{1}{2}\sqrt{\frac{height + width}{2}}$$

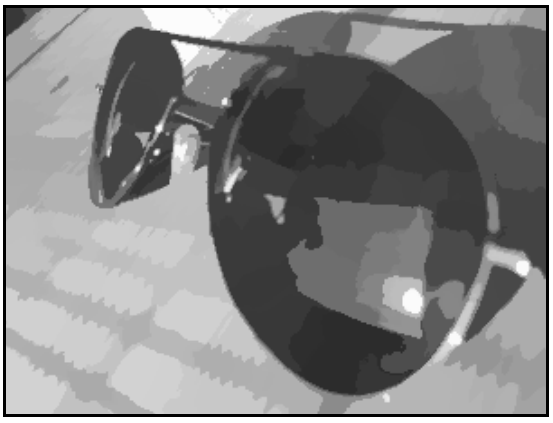
$$h = \sqrt{\frac{height + width}{2}}$$

Original Image:

$h = 10$



$h = 15$



$h = 20$

4. Edge Detection using Mean Shift Smoothing

The difference mean shift smoothing makes in the edge detection task can be observed by performing the same edge detection process on an image that has been smoothed, and comparing it with the outcome of the edge detector on the untouched image. The discussion about whether mean shift smoothing improves edge detection in the general case remains open, since there is no clear rule describing the cases in which mean shift smoothing does not yield better results (as will be presented in the following section). Nevertheless, it is safe to say that in many cases this smoothing process can indeed improve edge detection.

The edge detector implemented computes the inputted image's gradient amplitude estimation in the following manner:

$$\left|\nabla\right|(i,j)=\sqrt{\left(\frac{I(i,j+1)-I(i,j-1)}{2}\right)^2+\left(\frac{I(i+1,j)-I(i-1,j)}{2}\right)^2} \quad (8)$$

Two threshold values are required for the edge detection process: T_1 and T_2. The following hysteresis process was then applied to the gradient amplitude estimate, using these threshold values:

Given: gradient amplitude estimation $\left|\nabla\right|(i,j)$.

- Initialize all pixels of the edge map E as unlabeled.
- For each unlabeled pixel p:
  - If $\left|\nabla\right|_p > T_2$:
    Set E_p = EDGE;
    For each immediate neighbor q of p:
    If $\left|\nabla\right|_q > T_1$ set E_q = EDGE;
  - Else set E_p = NONEDGE;

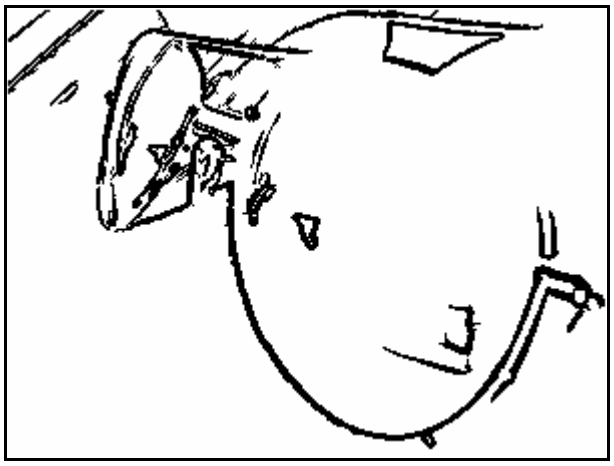
  Till no remaining unlabeled pixels.

Output: Edge Map $E(i,j)$.

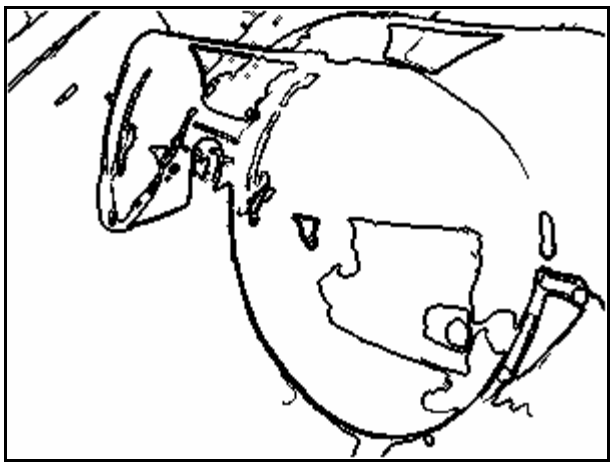
This hysteresis algorithm causes the edge detector to "follow" the real image edges, resulting in more continuous lines in the outputted edge map. The two parameters T_1 and T_2 can be modified to any desired value, while their default value is set to be 17 and 25, respectively.

In the following images, the effect of the mean shift smoothing in the edge detection task is presented. The edge detection is performed on both the smoothed and untouched images, to better notice the significance of the smoothing processes.

1. glasses



No Smoothing

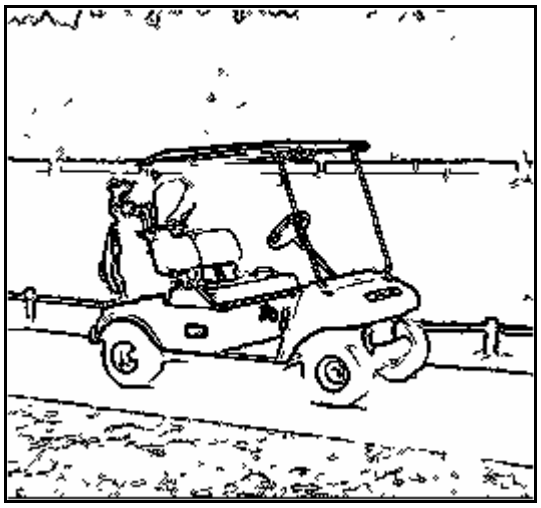


Smoothed

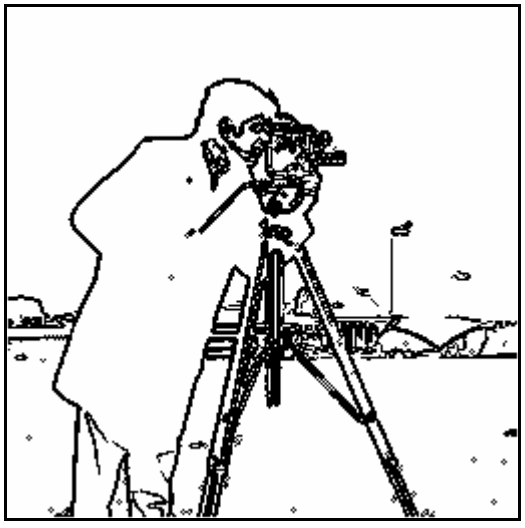
2. camera man

3. golf cart

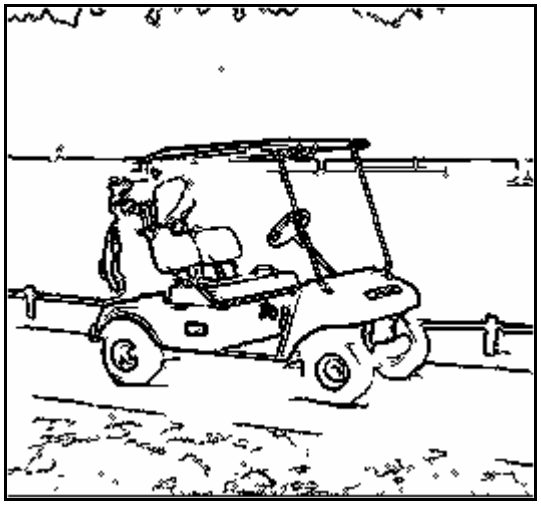
No Smoothing



No Smoothing

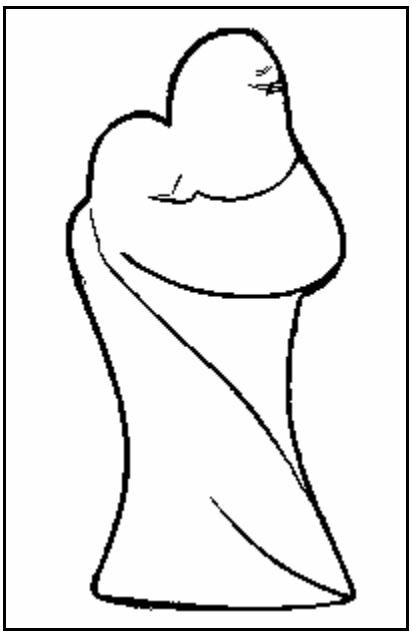


Smoothed

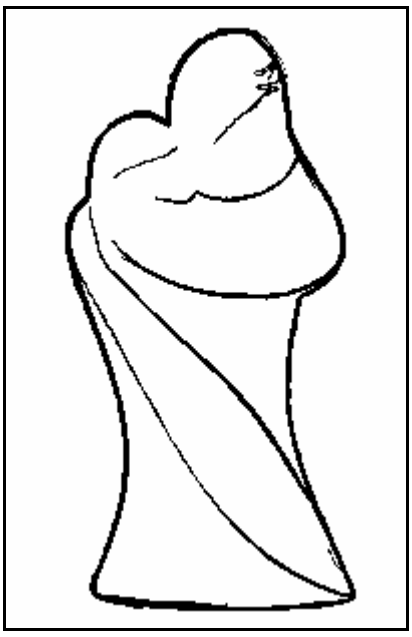


Smoothed

4. sculpture

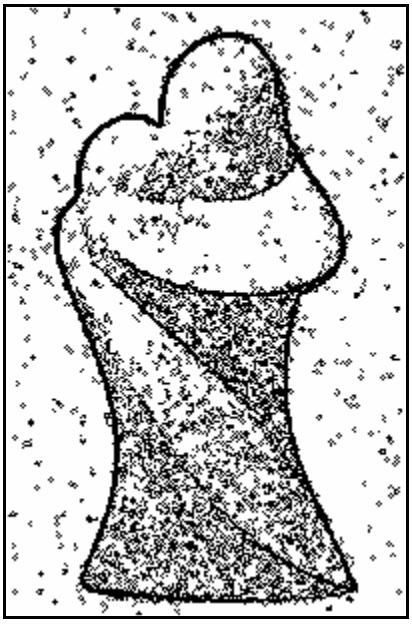


No Smoothing



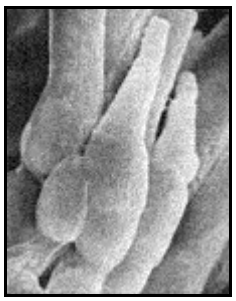
Smoothed

5. noisy sculpture



No Smoothing

Smoothed

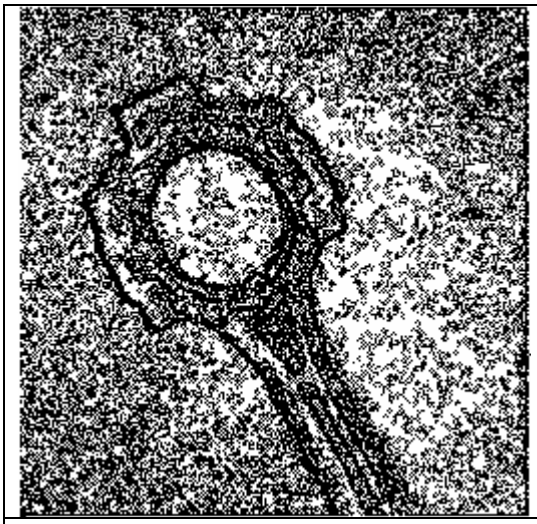


6. noisy cones

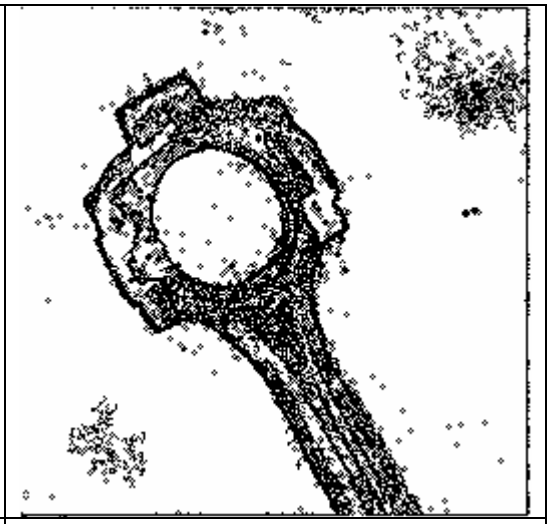
No Smoothing

Smoothed

7. noisy object



No Smoothing



Smoothed

5. Discussion

The initial idea for this project was a simpler implementation of an edge detector using a clustering algorithm: The data points were to be clustered and assigned labels. Pixels were to be set as edges if their corresponding data points were located on the boundary of two or more clusters, i.e. if a pixel has a different label than one or more of its neighbors, it is set to be an edge pixel. This policy did not prove itself as useful for two main reasons: It has almost no ability to handle objects that have strong variance in their shading. Furthermore, for entire objects to be labeled as one cluster, very large values of the parameter h are required, which has a major influence on running time – due to the scanning of a square range of $h \times h$ for each pixel in the clustering process ($O(h^2)$ per iteration, per pixel).

After this policy was tested and found to be unsuccessful, the adjustment to only smoothing the image using the mean shift clustering iteration was made. As one can notice in the images presented in the previous section, the mean shift smoothing can lead to good results in cases in which Gaussian noise is present (cases 5, 6 and 7), as well as in cases in which the smoothing amplifies the contrast along edges (cases 1 and 4). However, it is possible that the smoothing somewhat deteriorates the edge detection, as noticeable in case 3. This may happen when the smoothing process shifts the edge data points to the same areas as the points residing at both sides of the edge, reducing the contrast variance across the edge.

A major drawback of this edge detection method, as in many other edge detection methods, is the dependency of the edge detection quality on parameters inputted by the user: the window radius h, and the threshold values T_1 and T_2. Small modifications in the values of these parameters can produce drastic changes in the produced edge map. The value h can be computed as a function of the scale of the problem, which results in reasonable edge maps, nevertheless controlling the h value manually remains the best way to obtain optimal results. The implementation also defines default values for T_1 and T_2, but in some cases the user would prefer to change these values to improve the edge detection results.

It seems as if the more parameters needed to be inputted beforehand by the user in order to perform a computation, i.e. the more subjective judgment (by human interpretation of the problem) is needed, the weaker the computation is in terms of usefulness. Clearly, the goal of edge detector designers, or for this matter, whoever deals with computational vision problems, should be minimizing the need of subjective interventions to perform a task. In this context, the edge detector presented in this paper is inferior to some other well known edge detectors.

Various improvements of the edge detection method described in this paper might be possible. Some ideas for improvement:

- Dynamically choosing threshold values – A better way to determine the best values for the thresholds T_1 and T_2 might be breaking down the image to regions, according to some criteria (which will obviously require some computational

work), and producing the threshold values as a function of the gradient amplitude estimation in that region. If completed successfully, this can significantly reduce the level of intervention needed by the user for this edge detection method.

- Manipulation of the rescaling process – there might very well be a point in changing the policy of the rescaling of the vectors in the data point space. For this project, it was decided to attempt to create a data point set that has no preference to a specific component of the vectors, as much as possible. This was achieved by rescaling the intensity component to the scale of the height and width of the original image (see (5), section 2). It might be interesting to let the user set the desired scale for the intensity component. The shifting of each data point to high density regions is determined by considering proximity in both location and intensity. There is a delicate tradeoff between the two, which can be settled by manipulation of the rescaling stage. This will, of course, require even more parameters to be inputted manually, which is generally undesirable. However, this additional control might yield better results in the edge detection following the smoothing process.

6. References

[1] D. Comaniciu, P. Meer: *Distribution Free Decomposition of Multivariate Data*, (Invited), Pattern Analysis and Applications, Vol. 2, 22-30, 1999

[2] Lecture Notes from "Introduction to Computational and Biological Vision" at: http://www.cs.bgu.ac.il/~ben-shahar/Teaching/Computational-Vision/LectureNotes.php.