Survey Paper on Comparison of Image Segmentation Algorithm Techniques

Dr.Sharanabasava Inamadar
Dr. D Y Patil school of Engineering and technology Lohegaon Pune

***Abstract* - Image segmentation refers to the process of partitioning a Digital image into multiple segments (sets of pixels, also known as superpixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. In this survey paper we will compare the sobel, laplician and canny edge detection algorithm.**

***Index Terms* - Smart Stick, Arduino Uno, Location Tracking, Sensors, Object Detection.**

I.INTRODUCTION

Edge detection is a terminology in image processing and computer vision, particularly in the areas of feature detection and feature extraction, to refer to algorithm which aims at identifying points in a digital image at which the image brightness changes sharply or more formally have discontinuities.

Applying edge detection to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image. The image quality reflects significant information in the output edge and the size of the image is reduced. This in turn explains further that edge detection is one of the ways of solving the problem of high volume of space images occupy in the computer memory. The problems of storage, transmission over the Internet and bandwidth could be solved when edges are detected (Vincent, 2007). Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation.

They are various algorithms are available for the image edge detection. Some of them are as follows: Sowavelet based edge detection, canny edge detection, bel edge detection, computational approach, edge detection convolution, laplician edge detection algorithm Threshold edge detection, Gaussian edge detection, horizontal edge detection. In this paper we are planning to compare the performance techniques for some of the edge detection which are mentioned above.

II. LITERATURE SURVEY

1 The Canny Edge Detection Algorithm

The algorithm runs in 5 separate steps:

1. Smoothing: Blurring of the image to remove noise.
2. Finding gradients: The edges should be marked where the gradient of the image has large magnitudes.
3. non-maximum suppression: Only local maxima should be marked as edges.
4. 4.Double thresholding: Potential edges are determined by thresholding.
5. Edge tracking by hysteresis: Final edges are determined by suppressing all edges that are not connected to a very certain (strong) edge.

Each step is described in the following subsections.

2.1 Smoothing:

It is inevitable that all images taken from a camera will contain some amount of noise. To prevent that noise is mistaken for edges, noise must be reduced. Therefore, the image is first smoothed by applying a Gaussian filter. The kernel of a Gaussian filter with a standard deviation of _ = 1.4 is shown in Equation (1). The effect of smoothing the test image with this filter is shown in Figure 2.

Figure 1: The image used as example of Canny edge detection.

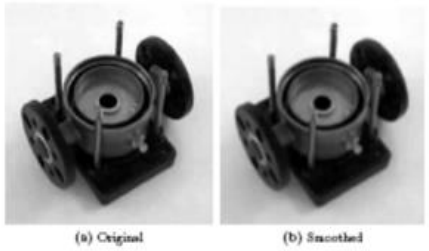


Figure 2: The original grayscale image is smoothed with a Gaussian filter to suppress noise.

2.2 Finding gradients

The Canny algorithm basically finds edges where the grayscale intensity of the image changes the most. These areas are found by determining gradients of the image. Gradients at each pixel in the smoothed image are determined by applying what is known as the Sobel-operator. First step is to approximate the gradient in the x- and y-direction respectively by applying the kernels shown in Equation (2)

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (2)$$

The gradient magnitudes (also known as the edge strengths) can then be determined as a Euclidean distance measure by applying the law of Pythagoras as shown in Equation (3). It is sometimes simplified by applying Manhattan distance measure as shown in Equation (4) to reduce the computational complexity. The Euclidean distance measure has been applied to the test image. The computed edge strengths are compared to the smoothed image in Figure 3.

$$|G| = \sqrt{G_x^2 + G_y^2} \quad (3)$$

$$|G| = |G_x| + |G_y| \quad (4)$$

where: Gx and Gy are the gradients in the x- and y-directions respectively. It is obvious from Figure 3, that an image of the gradient magnitudes often indicate the edges quite clearly. However, the edges are typically broad and thus do not indicate exactly where edges are. To make it possible to determine this (see Section 2.3), the direction of the edges must be determined and stored as shown in Equation (5).

$$\theta = \arctan\left(\frac{|G_y|}{|G_x|}\right) \quad (5)$$

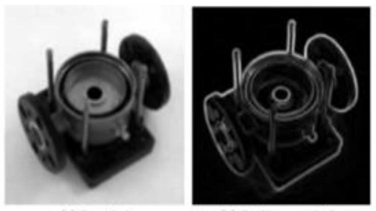


Figure 3: The gradient magnitudes in the smoothed image shown in 3b as well as their directions are determined by applying e.g. the Sobel-operator.

2.3 Non-maximum suppression

The purpose of this step is to convert the “blurred” edges in the image of the gradient magnitudes to “sharp” edges. Basically this is done by preserving all local maxima in the gradient image, and deleting everything else. The algorithm is for each pixel in the gradient image:

1. Round the gradient direction _ to nearest 45◦, corresponding to the use of an 8-connected neighbourhood.
2. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction. I.e. if the gradient direction is north (theta = 90◦), compare with the pixels to the north and south.
3. If the edge strength of the current pixel is largest; preserve the value of the edge strength. If not, suppress (i.e. remove) the value.

A simple example of non-maximum suppression is shown in Figure 4. Almost all pixels have gradient directions pointing north. They are therefore compared with the pixels above and below. The pixels that turn out to be maximal in this comparison are marked with white borders. All other pixels will be suppressed. Figure 5 shows the effect on the test image.

2.4 Double thresholding

The edge-pixels remaining after the non-maximum suppression step are (still) marked with their strength pixel-by-pixel. Many of be caused by noise or color variations for instance due to rough surfaces. The simplest way to discern between these would be to use a threshold, so that only edges stronger that a certain value would be preserved. The Canny edge detection algorithm uses double thresholding. Edge pixels stronger than the high threshold are marked as strong; edge pixels weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak. The effect on the test image with thresholds of 20 and 80 is shown in Figure 6.

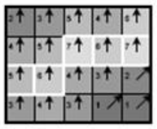
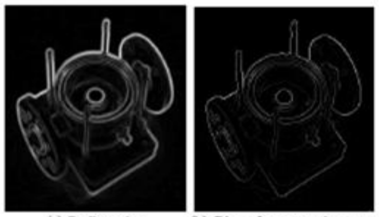
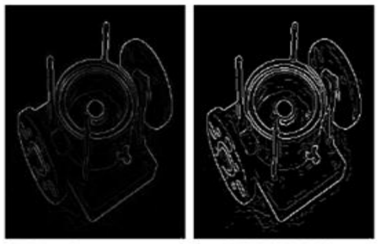


Figure 4: Illustration of non-maximum suppression. The edge strengths are indicated both as colors and numbers, while the gradient directions are shown as arrows. The resulting edge pixels are marked with white borders.



(a) Gradient values (b) Edges after non-maximum suppression

Figure 5: Non-maximum suppression. Edge-pixels are only preserved where the gradient has local maxima.

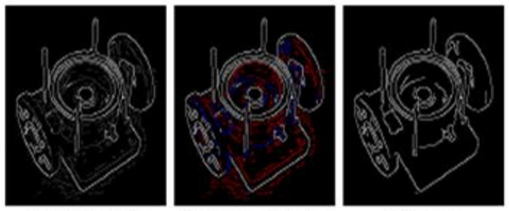


(a) Edges after non-maximum suppression (b) Double thresholding

Figure 6: Thresholding of edges. In the second image strong edges are white, while weak edges are grey. Edges with a strength below both thresholds are suppressed.

2.5Edge tracking by hysteresis

Strong edges are interpreted as "certain edges" and can immediately be included in the final edge image. Weak edges are included if and only if they are connected to strong edges. The logic is of course that noise and other small variations are unlikely to result in a strong edge (with proper adjustment of the threshold levels). Thus, strong edges will (almost) only be due to true edges in the original image. The weak edges can either be due to true edges or noise/color variations. The latter type will probably be distributed independently of edges on the entire image, and thus only a small amount will be located adjacent to strong edges. Weak edges due to true edges are much more likely to be connected directly to strong edges. Edge tracking can be implemented by BLOB-analysis (Binary Large Object). The edge pixels are divided into connected BLOB's using 8-connected neighbourhood. BLOB's containing at least one strong edge pixel are then preserved, while other BLOB's are suppressed. The effect of edge tracking on the test image is shown in Figure 7.



(a) Double thresholding (b) Edge tracking by hysteresis (c) Final output

Figure 7: Edge tracking and final output. The middle image shows strong edges in white, weak edges connected to strong edges in blue, and other weak edges in red.

Implementation of Canny Edge Detection

As noted in Section 1, all images in this worksheet (except the original) are produced by our implementation. A few things should be noted with regards to this:

1. The (source) image and the thresholds can be chosen arbitrarily.
2. Only a smoothing filter with a standard deviation of _ = 1.4 is supported (the one shown in Equation 1).
3. The implementation uses the "correct" Euclidean measure for the edge strengths, described in Section 2.2.
4. The different filters cannot be applied to edge pixels. This causes the output image to be 8 pixels smaller in each direction.

The last step in the algorithm known as edge tracking can be implemented as either iterative or recursive BLOB analysis [4]. A recursive implementation can use the grass-fire algorithm However, our implementation uses the iterative approach. First all weak edges are scanned for neighbour edges and joined into groups. At the same time it is marked which groups are adjacent. Then all of these markings are examined to determine which groups of weak edges are
connected to strong edges (directly or indirectly). All weak edges that are connected to strong edges are marked as strong edges themselves. The rest of the weak edges are suppressed. This can be interpreted as BLOB analysis where only BLOB's containing strong edges are preserved (and considered as one BLOB). Figure 8 shows the complete edge detection process on the test image including all intermediate results.

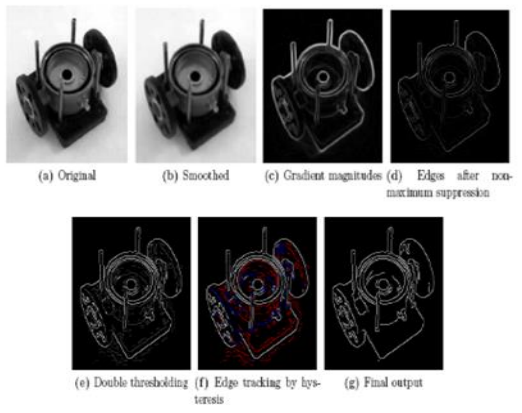


Figure 8: All steps of the edge detection.

|III SOBEL EDGE DETECTION ALGORITHM

Sobel Filter Design

Most edge detection methods work on the assumption that the edge occurs where there is a discontinuity in the intensity function or a very steep intensity gradient in the image. Using this assumption, if one take the derivative of the intensity value across the image and find points where the derivative is maximum, then the egde could be located. The gradient is a vector, whose components measure how rapid pixel value are changing with distance in the *x* and *y* direction. Thus, the components of the gradient may be found using the following approximation:

$$\frac{\partial f(x,y)}{\partial x} = \Delta x = \frac{f(x+dx,y)-f(x,y)}{dx} \quad (3)$$

$$\frac{\partial f(x,y)}{\partial x} = \Delta y = \frac{f(x,y+dy)-f(x,y)}{dy} \quad (4)$$

where *dx* and *dy* measure distance along the *x* and *y* directions respectively. In discrete images, one can consider *dx* and *dy* in terms of numbers of pixel between two points.

$dx = dy = 1$

*(*pixel spacing) is the point at which pixel coordinates areΔ *i*, *j* thus,

$$\Delta x = f(i+1,j) - f(i,j) \quad (5)$$

$$\Delta y = f(i,j+1) - f(i,j) \quad (6)$$

In order to detect the presence of a gradient discontinuity, one could calculate the change in the gradient at Δ *i*, *j* .This can be done by finding the following magnitude measure

$$M = \sqrt{\Delta x^2 + \Delta^2 y} \quad (7)$$

and the gradient direction is given by

$$\theta = \tan^{-1}\left[\frac{\Delta y}{\Delta x}\right] \quad (8)$$

IV. METHODOLOGY

There are many methods of detecting edges; the majority of different methods may be grouped into these two categories:

i.Gradient: The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. For example Roberts, Prewitt, Sobel where detected features have very sharp edges. (see Figure 1)

ii. Laplacian: The Laplacian method searches for zero crossings in the second derivative of the image to find edges e.g. Marr-Hildreth, Laplacian of Gaussian *etc*. An edge has one dimensional

shape of a ramp and calculating the derivative of the image can highlight its location (see Figure 2). Edges may be viewpoint dependent: these are edges that may change as the viewpoint changes and typically reflect the geometry of the scene which in turn reflects the properties of the viewed objects such as surface markings and surface shape. A typical edge might be the border between a block of red colour and a block of yellow, in contrast. However, what happens when one looks at the pixels of that image is that all visible portion of one edge is compacted.

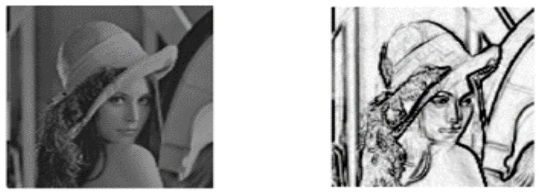


Figure 1: The Gradient Method

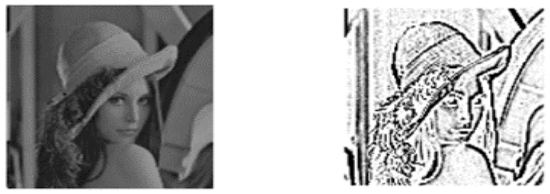


Figure 2: the Laplacian Method

The Sobel operator is an example of the gradient method. The Sobel operator is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function (Sobel & Feldman, 1968). The different operators in eq. (5) and (6) correspond to convolving the image with the following marks

$$\Delta x = \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, \Delta y = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$$

When this is done, then:
i. The top left-hand corner of the appropriate mask is super-imposed over each pixel of the image in turn,
ii. A value is calculated for Δx or Δy by using the mask coefficients in a weighted sum of the value of pixels Δ *i, j* □□and its neighbours,
iii. These masks are referred to as convolution masks or sometimes c of finding approximate gradient components along the *x* and *y* directions, approximation of the gradient components could be done along directions at 45 and 135□□to the axes respectively. In this case

$$\Delta x = f(i+1, j+1) - f(i, j) \quad (9)$$
$$\Delta y = f(i, j+1) - f(i+1, j) \quad (10)$$

This form of operator is known as the Roberts edge operator and was one of the first set of operators used to detect edges in images (Robert, 1965). The corresponding convolution masks are given by:

$$\Delta_1 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \text{ and } \Delta_2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

An advantage of using a larger mask size is that the errors due to the effects of noise are reduced by local averaging within the neighbourhood of the mask. An advantage of using a mask of odd size is that the operators are centered and can therefore provide an estimate that is based on a center pixel *(i,j)*. One important edge operator of this type is the Sobel edge operator. The Sobel edge operator masks are given as

$$\Delta x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \Delta y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point and therefore how likely it is that part of the image represents an edge, as well as how that the edge is likely to be oriented. In practice, the magnitude (likelihood of an edge) calculation is more reliable and easier to interpret than the direction calculation. Mathematically, the gradient of a two-variable function (the image intensity function) at each image point is a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points to the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction. This implies that the result of the Sobel operator at any image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values. The algorithm for developing the Sobel model for edge detection is given below.

```
Pseudo-codes for Sobel edge detection method
Input: A Sample Image
Output: Detected Edges
Step 1: Accept the input image
Step 2: Apply mask Gx, Gy to the input image
Step 3: Apply Sobel edge detection algorithm and the gradient
Step 4: Masks manipulation of Gx, Gy separately on the input image
Step 5: Results combined to find the absolute magnitude of the gradient
    $|G| = \sqrt{Gx^2 + Gy^2}$
Step 6: the absolute magnitude is the output edges
```

Second order derivative operators
A maximum of the first derivative will occur at a zero crossing of the second derivative. To get both horizontal and vertical edges, we look at second derivative in both the *x* and *y* directions.
This is the Laplacian of *I* where

$$\nabla^2 \mathrm{I} = \frac{\partial^2 \mathrm{I}}{\partial x^2} + \frac{\partial^2 \mathrm{I}}{\partial y^2}$$

The Laplacian is linear and rotationally symmetric. Thus, if one search for the zero crossing of the image that is first smoothed with a Gaussian mask and then the second derivative is calculated; or one can convolve the image with the Laplacian of the Gaussianalso known as the LoG operator.

$$\nabla^2 (G \otimes \mathrm{I}) = \nabla^2 G \otimes \mathrm{I}$$

The edge is modeled by specifying its four degrees of freedom: its position, its orientation, and the constant intensities on either side of the step. The data is matched by seeking the least squares error fit of the parametric model to the image window but such an approach is generally and computationally expensive. Normally what is done is that both the image data and the model are represented over small windows by their first derivative coefficients in a particular *2-D* orthonormal series expansion. In this case the optimization reduces to just one variable: the orientation of the edge.

V THE LAPLACIAN OF GAUSSIAN EDGE DETECTION ALGORITHM

The Laplacian of Gaussian(LoG) operator was used to locate the edges in the images. The edges were located by noting the zero crossings of the convolution of the LoG operator with the image. The LoG operator always creates closed contour edges; this is sometimes undesirable because it can lead to an edge at locations where there is only a slight variation in pixel intensity. The output of the edge detector is a black and white (binary) image showing the located edges. Figure 1 is the original `Lenna' image and the edges of the original `Lenna' image as detected by the LoG operator. Figure 2 is the original `SF' image and the edges of the original `SF' image as detected by the LoG operator.

Edge Distortion Results
Both the `Lenna' and `SF' images were vector quantized using the FSCL VQ algorithm. Each image was vector quantized using codebooks of sizes 32 and 128. The codebooks were created using a set of training images and were tested on two images not in the training set (`Lenna' and `SF').
Codebooks of both sizes were trained using both Euclidean distance and absolute distance as the distortion measure. The images were then vector quantized using the codebooks created. During the actual vector quantization of the images both the Euclidean distance and absolute distance distortion measures were used to pick the \closest" vector from the codebook. This results in 12 possible different vector quantized images. Table 1 summarizes the results. The number of pixels that differed between the reconstructed edge image and the original edge image were counted, and the percentage of different pixels as defined by Equation (4), is given in the Table.

Number of Different Pixels
Percentage of Different Pixels =_ 10 (4)

Total Number of Pixels
Table 1 shows the percentage of different pixels for both images encoded with both 32 and 128 codewords using three different methods. The difference between the three methods lies in the distortion measure used for the encoding process. One set of trials was performed using the Euclidean distance distortion measure for both training the codebook and for selecting the closest codeword in the reconstruction process. Another set of trials was done using the absolute distance distortion measure for both training the codebook and for selecting the closest codeword in the reconstruction process. The set of trials used Euclidean distance for training the codebook and absolute distance for selecting the closest codeword during reconstruction. Static codebooks were used, meaning that the codebooks are trained before the image reconstruction takes place and are not updated during image reconstruction. If static codebooks are used, the only computation during image encoding is to the codeword that is the \closest" match to the vector of the original image. The time required for this calculation depends on the distortion measure. Obviously, we would prefer the use of absolute distance for the distortion measure because it can be computed faster in hardware than can Euclidean distance and is the motivation for the experiment using Euclidean distance for training and absolute distance for reconstruction.
The data indicates no significant loss of edge information if absolute distance is used as a distortion measure. Figure 3 (left) is the edge detector output for the `Lenna' image using the absolute distance distortion measure for quantization with 128 codewords. Figure 3 (right) compares the original `Lenna' edges and the absolute `Lenna' edges. The dark spots indicate where the edge images differ. The edge images and the comparison image show that none of the major edges of the original images are significantly modified in the vector quantized reconstructions
Table 1: Percentage of Different Pixels for Various Images and Methods

Image/Number of Codewords	Method	Percentage of Different Pixels
Lenna/32	Euclidean-Euclidean	11.449%

	Absolute-Absolute	11.767%
	Euclidean-Absolute	11.671%
SF/32	Euclidean-Euclidean	14.212%
	Absolute-Absolute	14.602%
	Euclidean-Absolute	14.578%
Lenna/128	Euclidean-Euclidean	8.817%
	Absolute-Absolute	8.974%
	Euclidean-Absolute	9.254%
SF/128	Euclidean-Euclidean	11.958%
	Absolute-Absolute	12.286%
	Euclidean-Absolute	12.248%

The data indicates no significant loss of edge information if absolute distance is used as a distortion measure. Figure 3 (left) is the edge detector output for the `Lenna' image using the absolute distance distortion measure for quantization with 128 codewords. Figure 3 (right) compares the original `Lenna' edges and the absolute `Lenna' edges. The dark spots indicate where the edge images differ. The edge images and the comparison image show that none of the major edges of the original images are significantly modified in the vector quantized reconstructions

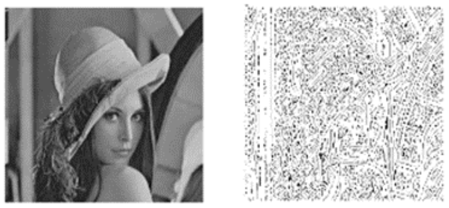


Figure 1: Original Lenna Image (left), Edges of Original Lenna Image (right)

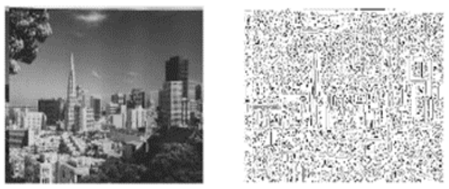


Figure 2: Original SF Image (left), Edges of Original SF Image (right)

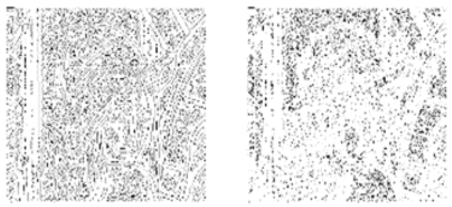


Figure 3: Edges of Reconstructed Lenna Using Absolute Distance and 128 Codewords (left), Comparison of Original Lenna Edges with the Edges of Reconstructed Lenna Using Absolute Distance and 128 Codewords (right)

VI. TILE SIZE

Tile size is another consideration when designing a vector quantizer. In this study tile sizes of 2 _ 2 (4 pixels), 3 _ 3 (9 pixels), and 4 _ 4 (16 pixels) were investigated to determine the advantages and disadvantages of various tile sizes. The FSCL VQ algorithm was used to design six different vector quantizer codebooks in which the tile size and number of codewords were varied for the different vector quantizers. The FSCL algorithm was used to generate the codebooks. The data used for training the codebooks was an image" formed by the concatenation of ve different images. The image was then broken into vectors for training. In this fashion the FSCL algorithm was used to create codebooks containing 2 _ 2, 3 _ 3, and 4 _ 4 vectors. For each tile size codebooks of 128 and 512 codewords were created. Testing images were the same as the training images. The images were reconstructed using the six different vector quantizers codebooks. For each codebook the MSE value and the bits per pixel (BPP) of the reconstructed image were recorded. Note that the codebooks are trained until the MSE value for the testing image becomes constant iterations (here an iteration

The MSE value as a function of the number of training iterations (here an iteration is one pass through the entire training ensemble) was also recorded

6.1 Tile Size Results

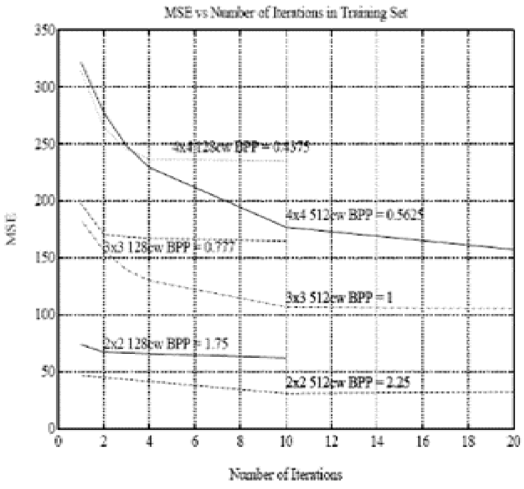


Figure 4: MSE vs Number of Iterations on the Training Set

MSE vs Number of Iterations on the Training Set Two graphs summarize the results of the tile size experiments. The_ first graph (Figure shows plot of MSE value a function of number of training iterations for the various combinations of tile size and number of codewords.

The second graph (Figure 5) shows the MSE value after training is completed as compared to the Bits Per Pixel.

BPP is a measure that is inversely proportional to compression. The number of bits per pixel for the original (non- vector-quantized) images is eight bits per pixel. The data indicates that as tile size increases the MSE value increases for a_ fixed number of codewords. As expected, the MSE value decreases as the number of codewords is increased .As the BPP is a function of both tile size and number of codewords, the BPP increases as number of codewords increases, and decreases as tile size increases. The MSE values and compression ratios are summarized in Table 2.Vector quantization with 2_2 tiles yield the lowest MSE value given a_fixed number of codewords. 2_2 tiles also require fewer iterations to reach the point where the MSE value levels o_. The BPP for the 2_2 tiles are the highest, meaning this yields the minimal compression. The images vector quantized with 2_2 tiles have the best edge detail and are less blocky than the images encoded with larger tile sizes. When 3_3 tiles are used the edges of the image are less definite and the block lines of the images are increased. Similarly, 4_4 tiles yield the highest MSE value, the lowest BPP (greatest compression) and require the greatest number of iterations. These images are very blocky, and the edges look quite poor. Based on these results we have restricted our attention to 2_2 tiles which provides reasonable compression rates, very good image quality, and permit hardware realizations with current technology

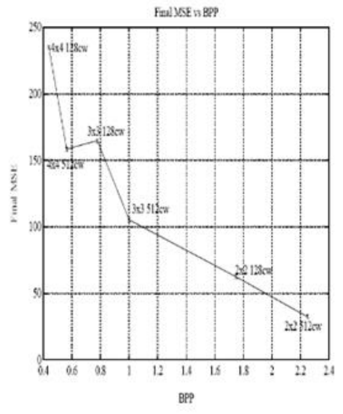


Figure 5: Final MSE vs BPP as a Function of Codebook Size and the Number of Codewords

VII. CONCLUSION

The algorithms presented successfully extracts edge-end pixels in their entirety. The simplicity of the proposed algorithm should make it an attractive tool for edge-based image segmentation, essential in biological cell image analysis and indeed in any image processing task.

REFERENCES

[1] Sergei Azernikov. “Sweeping solids on manifolds”. In Symposium on Solid and Physical Modeling, pages 249–255, 2008.

[2] John Canny. “A computational approach to edge detection. Pattern Analysis and Machine” Intelligence, IEEE Transactions on, PAMI-8(6):679–698, Nov. 1986.

[3] F. Mai, Y. Hung, H. Zhong, and W. Sze. “A hierarchical approach for fast and robust ellipse extraction. Pattern Recognition”, 41(8):2512–2524, August 2008.

[4] Thomas B. Moeslund. “Image and Video Processing”. August 2008

[5] Agaian, S. S., Baran, T. A., & Panetta, K. A. (2003). “Transform-based image compression by noise reduction and spatial modification using Boolean minimization”. *IEEE Workshop on Statistical Signal Processing*, 28 Sept.-1 Oct. pp. 226 – 229.

[6] Baker, S., & Nayar, S. K. (1996). Pattern rejection. “*Proceedings of IEEE Conference Computer Vision and Pattern Recognition”*, 544-549.

[7] Canny, J. F. (1986). “A computational approach to edge detection. *IEEE Trans Pattern Analysis and Machine” Intelligence, 8*(6), 679-698.

[8] Chang-Huang, C. (2002). “*Edge detection based on class ratio”*. 152, sec.3, Peishen Rd., Shenkeng, Taipei, 22202, Taiwan, R.O.C.

[9] Folorunso O., Vincent, O. R., & Dansu, B. M. (2007). “Image edge detection” A knowledge management technique for visual scene analysis. *Information Management and Computer Security, 15*(1), 23-32.

[10] Gonzalez, R., & Woods, R. (2002).”*Digital image processing* (2nd ed.)” Prentice-Hall Inc. 567-612.

[11] Guthe, S., & Strasser, W. (2004). “Advanced techniques for high-quality multi-resolution volume rendering” *Graphics and Visualization, 28*, 25-78.

[12] Keren, D., Osadchy, M., & Gotsman. C. (2001). “Antifaces: A novel, fast method for image

detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence", 23*(7), 747-761.

[13] Mario, D., & Maltoni, D. (1997). "Direct gray-scale minutiae detection in fingerprints. *IEEE Transactions on Pattern Analysis and Machine Intelligence", 19*(1), 27-40.

[14] Michael, U. (2003). "Mathematics properties of the JPEG 2000 wavelet filters". *IEEE Transactions on Image Processing, 12*(9), 080-1090.

[15] Milan, S., Vaclav, H., & Roger, B. (2002). "*Image processing analysis and machine vision*. London: Chapman and Hall, 255-280".

[16] Osuna, E., Freund, R., & Girosi, F. (1997). "Training support vector machines" An applicat ion to face detection. *Proceedings of IEEE Conference Computer Vision and Pattern Recognition*.

[17] Priotr, P. (2004). "Practical signal decomposition design based on Haar – Wavelet transformation. *Journal of Applied Computer Science", 2*(1), 61-82.

[18] Roberts, L. G. (1965). "Machine perception of three-dimensional solids. In J. T. Tippett (Ed.), *Optical and electro-optical information processing* (Ch. 9, pp. 159-197). Cambridge, Massachusetts": MIT Press.

[19] Scharr, H. (1996). "*Digitale Bildverarbeitung und Papier: Texturanalyse mittels Pyramiden und Grauwertstatistiken am Beispiel der Papierformation*." Master's thesis, FakultÄat fÄur Physik und Astronomie, Ruprecht-Karls-UniversitÄat Heidelberg, Germany.

[20] Sobel, I., & Feldman, G. (1968)." *A 3 × 3 isotropic gradient operators for image processing*." Presented at a talk at the Stanford Artificial Project.

[21] Sparr, G. (2002). "*Image processing and pattern classification for character recognition*. Center for Mathematical Sciences", Lund University, 2, 25-78.

[22] Vincent, O. R., Folorunso, O., & Agboola, A. A. A. (2006). "Modified algorithm for transform-based image" compression. *Journal of Computer Science and its Application, 13*(1), 43-48.

[23] Vincent, O. R. (2007)".Optimization of network bandwidth using image compression". *Proceedings of the National Conference on advanced Data Computing Communications and Security*, Kadi, India.

[24] Yuval, F. (1996). "*Fractal image compression (theory and application)".* Institute for non-linear Science, University of California, San Diego, USA. Vincent & Folorunso 107