Homework 0 : Alohamora!
CMSC733

Aditya Jadhav
Master of Engineering in Robotics
University of Maryland, College Park, 20740
amjadhav@umd.edu

***Abstract*—This document presents my submission of Homework 0 - Alohomora. The report contains a detailed solution to the Phase 1 - Shake My Boundary problem. The approach followed to solve the Boundary Detection problem with state-of-the-art algorithms is illustrated in [1].**

I. Introduction

One of the fundamental problems in Computer Vision is Boundary Detection. Considerable efforts have been made towards the solution of this problem with focuses on detection and localization of color/intensity discontinuities and dividing the image into homogeneous regions. One such approach in [1] with Contour Detection and Image Segmentation methods significantly outperforms established algorithms like Canny Edge and Sobel Descriptors. Making use of the Pb (Probability of Boundary) Detection algorithm in [1], the report presents a solution of Phase 1 - Shake My Boundary.

II. Phase - 1

This section describes the process followed to implement a simplified version of Pb Detection algorithm (Pb-Lite algorithm) and analyzes the solutions obtained. The process flow is as follows:

- Generating Filter Banks
- Texton, Brightness and Color Map Development $\mathcal{T}$, $\mathcal{B}$, $\mathcal{C}$
- Computation of Texture, Brightness and Color Gradients $\mathcal{T}_g$, $\mathcal{B}_g$, $\mathcal{C}_g$
- Combining the Gradient Result with weighted Canny and Sobel Baselines

A. Filter Bank Generation

As the first part of the process of Pb-Lite Detection, we use three different filter banks with various scales and orientations to obtain the low-level features of the images which helps measuring texture properties and aggregating regional texture and brightness distributions. We use the Oriented Derivative of Gaussian (DoG) filter bank, the Leung-Malik filter bank and the Gabor Filters as illustrated in figures 1, 2, 3.

1) Oriented Derivative of Gaussian Filters: The oriented Derivative of Gaussian (DoG) filters have the first order derivative of the Gaussian Function, which can be created by convolving a Sobel descriptor and a Gaussian kernel and then rotating the result. The filter bank in generated with three different scales of 4, 7 and 10 with 15 orientations spaced over 0 to 360 degrees.

2) Leung-Malik Filters: This filter bank is another set of multi scale, multi orientation filters of four different types totalling to 48 filters. It has 36 first and second order derivatives of the Gaussian filters, 8 Laplacian of Gaussian filters and 4 Gaussian filters to complete the bank.

3) Gabor Filters: The frequency and orientation representations of the Gabor filters are claimed to be similar to the Human Visual System. These Gabor filters are implemented by modulating a Gaussian kernel with a sinusoidal kernel. The Gabor filters analyze the regions of interest for image content that corresponds to a specific frequency λ. The filter bank is generated with three different scales of 9, 16 and 25 with 15 orientations spaced over 0 to 360 degrees.

The three filter banks are illustrated with the following figures:

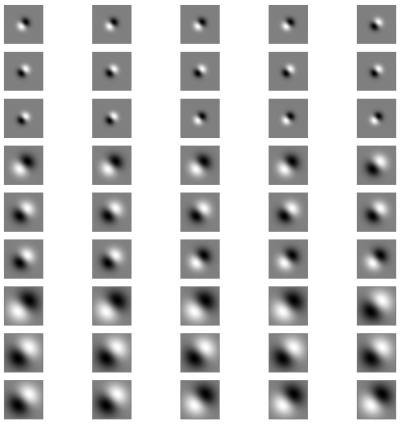

Fig. 1. The Derivative of Gaussians Filter Bank

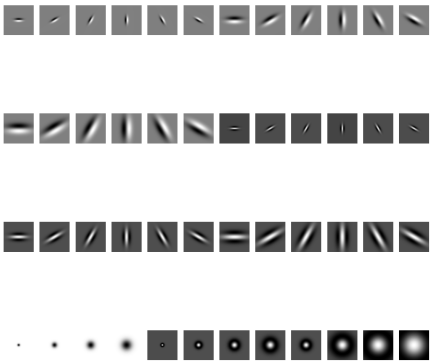

Fig. 2. The Leung-Malik Filter Bank

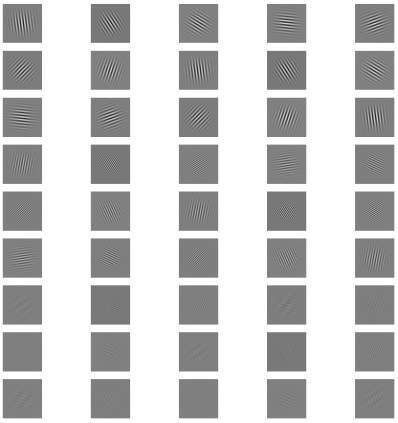

Fig. 3. The Gabor Filter Bank

B. Texton, Brightness and Color Map Development

In this section, we filter the input image with each filter of our filter banks. This filtering of the input image results in a vector of responses at each pixel. In our case, 186 filters create 186 filter responses at each pixel. The filter distributions are clustered into K textons with the use of K-means clustering algorithm. This dimensionality reduction of the filter distributions generates discreet Texton-IDs for each pixel. Each pixel is now replaced with a unique Texton-ID which creates the Texton map.

Similar to the Texton map development, brightness and color maps are generated. Intensity changes represented by Grayscale values are at the heart of the brightness map. K-means clustering is used to cluster intensity values into 16 clusters. This gives us the brightness map.

The color map represents the changes in color (RGB values) of the pixels using similar mechanisms as the brightness and texton map generation. These maps are illustrated by the following figures:

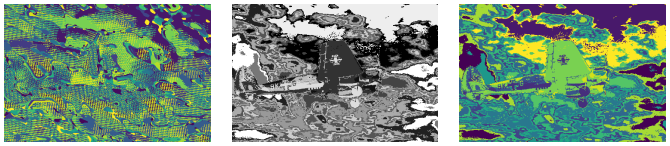

Fig. 4. Texture, Brightness and Color Maps of Image 1.png

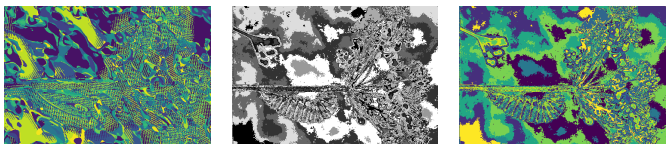

Fig. 5. Texture, Brightness and Color Maps of Image 2.png

Fig. 6. Texture, Brightness and Color Maps of Image 3.png

Fig. 7. Texture, Brightness and Color Maps of Image 4.png

Fig. 8. Texture, Brightness and Color Maps of Image 5.png

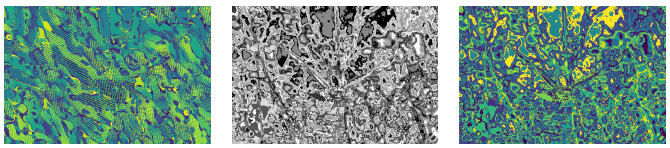

Fig. 9. Texture, Brightness and Color Maps of Image 6.png

Fig. 10. Texture, Brightness and Color Maps of Image 7.png

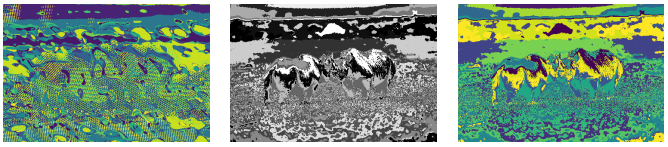

Fig. 11. Texture, Brightness and Color Maps of Image 8.png

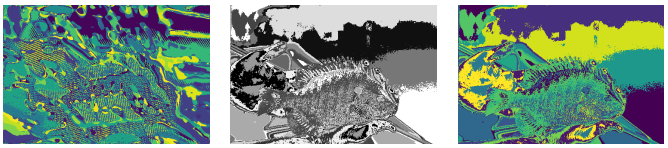

Fig. 12. Texture, Brightness and Color Maps of Image 9.png

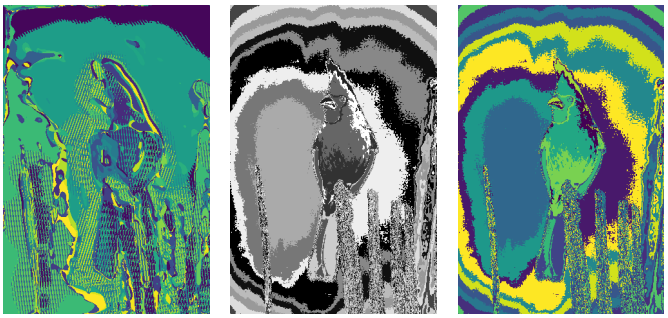

Fig. 13. Texture, Brightness and Color Maps of Image 10.png

C. Computation of Texture, Brightness and Color Gradients

The next step is to use the maps generated in the previous section to compute Texture, Brightness and Color gradients $\mathcal{T}_g$, $\mathcal{B}_g$, $\mathcal{C}_g$ which encode how much Texture, Brightness and Color distributions are changing at a pixel. Half Disk masks with multiple scales and orientations are employed for this step to calculate the gradients at each pixel. Half Disk masks are simply a set of binary semi-circular half disks concentric with the centre of the masks. To get the Texton, Brightness and Color distributions, the left/right half disk masks are applied at each pixel and the difference between them is calculated using the χ^2 (Chi-Square) distance given by:

$$\chi^2(g,h) = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i}$$

. The Half Disk Masks and the Gradient maps generated are shown below.

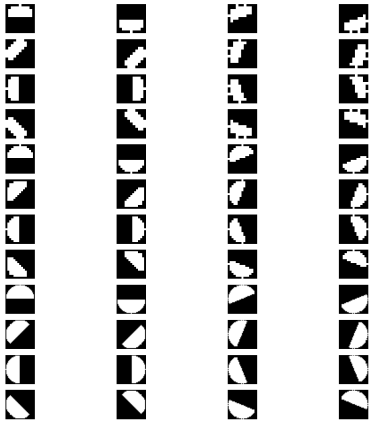

Fig. 14. Half Disk Masks with Radii 5, 7, 16

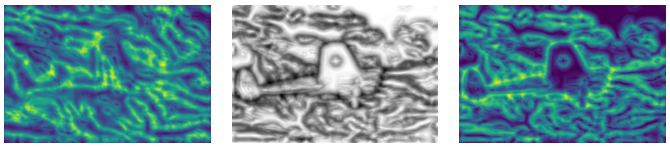

Fig. 15. Texture, Brightness and Color Gradients of Image 1.png

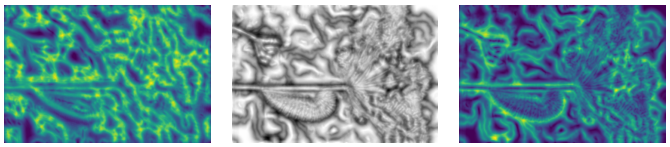

Fig. 16. Texture, Brightness and Color Gradients of Image 2.png

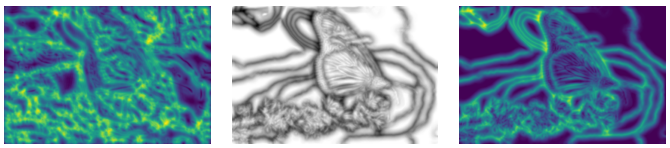

Fig. 17. Texture, Brightness and Color Gradients of Image 3.png

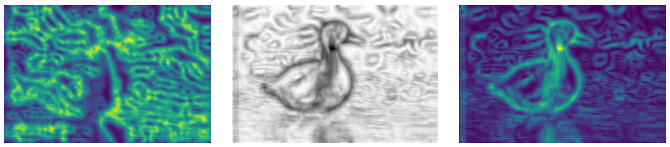

Fig. 18. Texture, Brightness and Color Gradients of Image 4.png

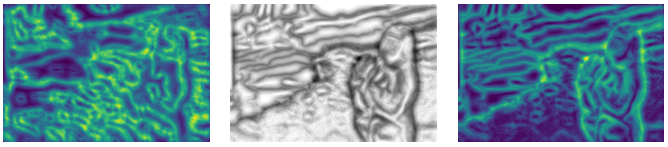

Fig. 19. Texture, Brightness and Color Gradients of Image 5.png

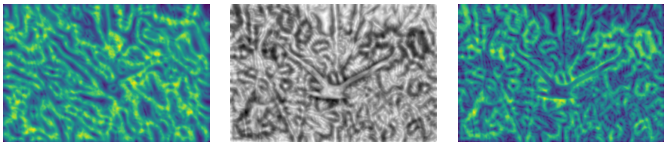

Fig. 20. Texture, Brightness and Color Gradients of Image 6.png

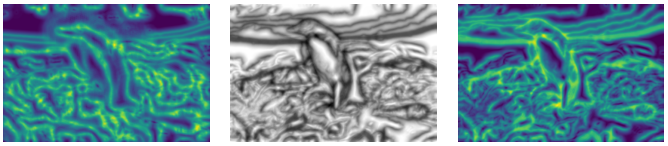

Fig. 21. Texture, Brightness and Color Gradients of Image 7.png

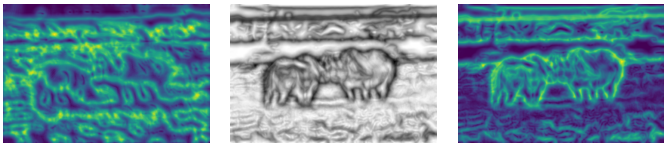

Fig. 22. Texture, Brightness and Color Gradients of Image 8.png

Fig. 23. Texture, Brightness and Color Gradients of Image 9.png

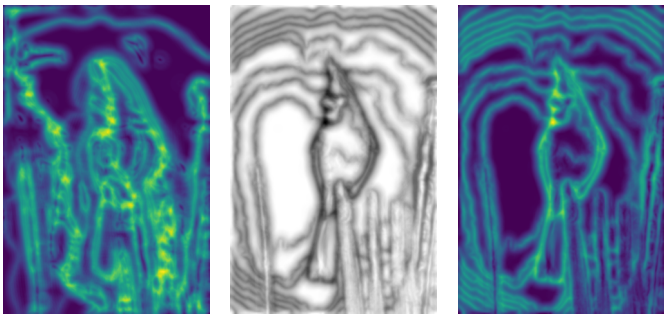

Fig. 24. Texture, Brightness and Color Gradients of Image 10.png

D. Combining Gradient Maps with Sobel, Canny Baselines

Finally, the Texture, brightness and Color gradients are averaged and added to form a feature vector using the following equation:

$$PbEdges = (\frac{T_g + B_g + C_g}{3}) \odot (w_1 * cannyPb + w_2 * sobelPb)$$

The Sobel and Canny baselines are used to obtain a weighted sum with weights w_1 , w_2. The feature vector and the weighted sum of baselines is added to get the final Pb-Lite output.

Fig. 25. Sobel, Canny and Pb-Lite Outputs of Image 1.png

Fig. 26. Sobel, Canny and Pb-Lite Outputs of Image 2.png

Fig. 27. Sobel, Canny and Pb-Lite Outputs of Image 3.png

Fig. 28. Sobel, Canny and Pb-Lite Outputs of Image 4.png

Fig. 29. Sobel, Canny and Pb-Lite Outputs of Image 5.png

Fig. 30. Sobel, Canny and Pb-Lite Outputs of Image 6.png

Fig. 31. Sobel, Canny and Pb-Lite Outputs of Image 7.png

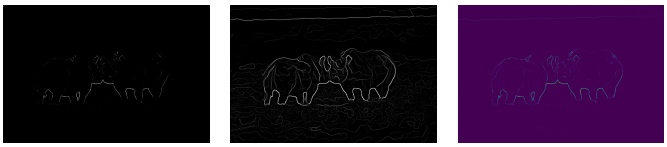

Fig. 32. Sobel, Canny and Pb-Lite Outputs of Image 8.png

Fig. 33. Sobel, Canny and Pb-Lite Outputs of Image 9.png

Fig. 34. Sobel, Canny and Pb-Lite Outputs of Image 10.png

III. Conclusion

The Sobel and Canny baselines produce an unnecessary amount of edges where the image contains intricate texture resulting in false positives. The Pb-Lite on the other hand does separate the textures and edges appropriately because the texture gradient can be controlled by the user. Therefore, Pb-Lite seems to be better at filtering out the noise to get edges. However, the Canny baseline manages to get sharper and accurate edges (though with noise) than the Pb-Lite outputs.

This was backed by the experimentation of weights in the Pb-Lite output. When the Canny baseline was given a lower weight (< 0.3) than the Sobel baseline (> 0.7), the Pb-Lite output could suppress noisy textures more accurately. Whereas, higher Canny baseline weights produced results with thicker edges along with unnecessary noise.

References

[1] P. Arbeláez, M. Maire, C. Fowlkes and J. Malik, "Contour Detection and Hierarchical Image Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 5, pp. 898-916, May 2011, doi: 10.1109/TPAMI.2010.161.

[2] Wikipedia - Gabor Filter

[3] CS 143 Brown University