Applied Mathematical Sciences, Vol. 2, 2008, no. 31, 1507 - 1520

Edge Detection Techniques:

Evaluations and Comparisons

Ehsan Nadernejad

Department of Computer Engineering, Faculty of Engineering
Mazandaran Institute of Technology
P.O. Box: 744, Babol, Iran
ehsan_nader@yahoo.com

Sara Sharifzadeh

Department of Communication Engineering, Faculty of Engineering
Shomal higher-education Institute,
P.O. Box: 731, Amol, Iran
sarasharifzade@yahoo.com

Hamid Hassanpour

Department of Computer Engineering, Faculty of Engineering
Mazandaran Institute of Technology
P.O. Box: 744, Babol, Iran
h_hassanpour@yahoo.com

Abstract

Edge detection is one of the most commonly used operations in image analysis, and there are probably more algorithms in the literature for enhancing and detecting edges than any other single subject. The reason for this is that edges form the outline of an object. An edge is the boundary between an object and the background, and indicates the boundary between overlapping objects. This means that if the edges in an image can be identified accurately, all of the objects can be located and basic properties such as area, perimeter, and shape can be measured. Since computer vision involves the identification and classification of objects in an image, edge detections is an essential tool. In this paper, we have compared several techniques for edge detection in image processing. We consider various well-known measuring metrics used in image processing applied to standard images in this comparison.

Keywords: image processing, edge detection, Euclidean distance, canny detector

I. INTRODUCTION

Edge detection is a very important area in the field of Computer Vision. Edges define the boundaries between regions in an image, which helps with segmentation and object recognition. They can show where shadows fall in an image or any other distinct change in the intensity of an image. Edge detection is a fundamental of low-level image processing and good edges are necessary for higher level processing. [1]

The problem is that in general edge detectors behave very poorly. While their behavior may fall within tolerances in specific situations, in general edge detectors have difficulty adapting to different situations. The quality of edge detection is highly dependent on lighting conditions, the presence of objects of similar intensities, density of edges in the scene, and noise. While each of these problems can be handled by adjusting certain values in the edge detector and changing the threshold value for what is considered an edge, no good method has been determined for automatically setting these values, so they must be manually changed by an operator each time the detector is run with a different set of data.

Since different edge detectors work better under different conditions, it would be ideal to have an algorithm that makes use of multiple edge detectors, applying each one when the scene conditions are most ideal for its method of detection. In order to create this system, you must first know which edge detectors perform better under which conditions. That is the goal of our project. We tested four edge detectors that use different methods for detecting edges and compared their results under a variety of situations to determine which detector was preferable under different sets of conditions. This data could then be used to create a multi-edge-detector system, which analyzes the scene and runs the edge detector best suited for the current set of data. For one of the edge detectors we considered two different ways of implementation, one using intensity only and the other using color information.

We also considered one additional edge detector which takes a different philosophy to edge detection. Rather than trying to find the ideal edge detector to apply to traditional photographs, it would be more efficient to merely change the method of photography to one which is more conducive to edge detection. It makes use of a camera that takes multiple images in rapid succession under different lighting conditions. Since the hardware for this sort of edge detection is different than that used with the other edge detectors, it would not be included in the multiple edge detector system but can be considered as a viable alternative to this.

II. REVIEW OF EDGE DETECTOR

A. The Marr-Hildreth Edge Detector

The Marr-Hildreth edge detector was a very popular edge operator before Canny released his paper. It is a gradient based operator which uses the Laplacian to take the second derivative of an image. The idea is that if there is a step difference in the intensity of the image, it will be represented by in the second derivative by a zero crossing:

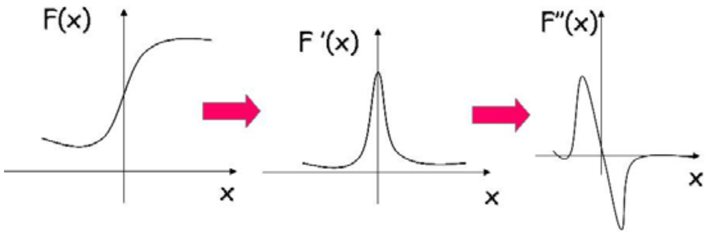


Figure.1

So the general algorithm for the Marr-Hildreth edge detector is as follows:

1. Smooth the image using a Gaussian. This smoothing reduces the amount of error found due to noise.

2. Apply a two dimensional Laplacian to the image:

$$r^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{1}$$

This Laplacian will be rotation invariant and is often called the "Mexican Hat operator" because of its shape:

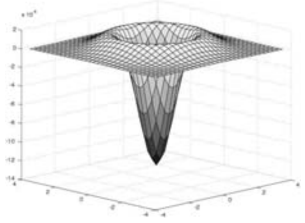


Figure.2

This operation is the equivalent of taking the second derivative of the image.

3. Loop through every pixel in the Laplacian of the smoothed image and look for sign changes. If there is a sign change and the slope across this sign change is greater than some threshold, mark this pixel as an edge. Alternatively, you can run these changes in slope through a hysteresis (described in the Canny edge detector) rather than using a simple threshold.

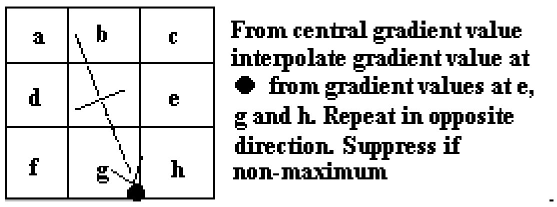
(Algorithm taken from [6])

B. The Canny Edge Detector

The Canny edge detector is widely considered to be the standard edge detection algorithm in the industry. It was first created by John Canny for his Masters thesis at MIT in 1983 [2], and still outperforms many of the newer algorithms that have been developed. Canny saw the edge detection problem as a signal processing optimization problem, so he developed an objective function to be optimized [2]. The solution to this problem was a rather complex exponential function, but Canny found several ways to approximate and optimize the edge-searching problem. The steps in the Canny edge detector are as follows:

1. Smooth the image with a two dimensional Gaussian. In most cases the computation of a two dimensional Gaussian is costly, so it is approximated by two one dimensional Gaussians, one in the x direction and the other in the y direction.

2. Take the gradient of the image. This shows changes in intensity, which indicates the presence of edges. This actually gives two results, the gradient in the x direction and the gradient in the y direction.
3. Non-maximal suppression. Edges will occur at points the where the gradient is at a maximum. Therefore, all points not at a maximum should be suppressed. In order to do this, the magnitude and direction of the gradient is computed at each pixel. Then for each pixel check if the magnitude of the gradient is greater at one pixel's distance away in either the positive or the negative direction perpendicular to the gradient. If the pixel is not greater than both, suppress it.



From central gradient value interpolate gradient value at ● from gradient values at e, g and h. Repeat in opposite direction. Suppress if non-maximum

Figure.3

4. Edge Thresholding. The method of thresholding used by the Canny Edge Detector is referred to as "hysteresis". It makes use of both a high threshold and a low threshold. If a pixel has a value above the high threshold, it is set as an edge pixel. If a pixel has a value above the low threshold and is the neighbor of an edge pixel, it is set as an edge pixel as well. If a pixel has a value above the low threshold but is not the neighbor of an edge pixel, it is not set as an edge pixel. If a pixel has a value below the low threshold, it is never set as an edge pixel.
(Algorithm based on description given in [3])

C. *The Local Threshold and Boolean Function Based Edge Detection* [1]

This edge detector is fundamentally different than many of the modern edge detectors derived from *Canny's* original. It does not rely on the gradient or Gaussian smoothing. It takes advantage of both local and global thresholding to find edges. Unlike other edge detectors, it converts a window of pixels into a binary pattern based on a local threshold, and then applies masks to determine if an edge exists at a certain point or not. By calculating the threshold on a per pixel basis, the edge detector should be less sensitive to variations in lighting throughout the picture. It does not rely on blurring to reduce noise in the image. It instead looks at the variance on a local level. The algorithm is as follows:

1. Apply a local threshold to a 3x3 window of the image. Because this is a local threshold, it is recalculated each time the window is moved. The threshold value is calculated as the mean of the 9 intensity values of the pixels in the window minus some small tolerance value. If a pixel has an intensity value greater than this threshold, it is set to a 1. If a pixel has an intensity value less than this threshold, it is set to a 0. This gives a binary pattern of the 3x3 window.

2. Compare the binary pattern to the edge masks. There are sixteen possible edge-like patterns that can arise in a 3x3 window, as shown below:

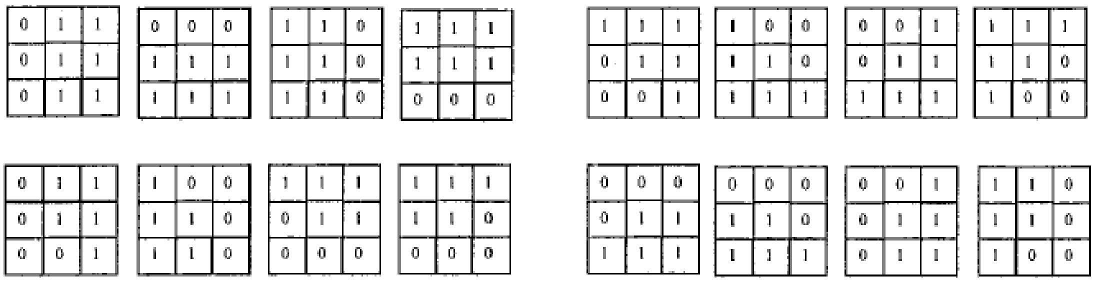


Figure.4

If the binary window obtained in step 1 matches any of these sixteen masks, the center pixel of the window is set to be an edge pixel.
3. Repeat steps 1 and 2 for each pixel in the image as the center pixel of the window. This will give all edges, but it will also give some false edges as a result of noise.
4. Use a global threshold to remove false edges. The variance for each 3x3 window is calculated, which will have a maximum at an edge. This value is then compared with a global threshold based on the level of noise in the image. If the value is greater than the threshold, it is kept as an edge. If it is not greater than the threshold, it is removed.

D: Color Edge Detection Using Euclidean Distance and Vector Angle [4]

Most edge detectors work on the grayscale representation of the image. This cuts down the amount of data you have to work with (one channel instead of three), but you also lose some information about the scene. By including the color component of the image, the edge detector should be able to detect edges in regions with high color variation but low intensity variation.

This edge detector uses two operators: Euclidean Distance and Vector Angle. The Euclidean Distance is a good operator for finding edges based on intensity and the Vector Angle is a good operator for finding edges based on hue and saturation. The detector applies both operators to the RGB color space of an image, and then combines the results from each based on the amount of color in a region. There is a difference vector and a vector gradient version; we chose to implement the vector gradient version. The Euclidean Distance between two pixels is defined as:

$$D(\vec{v}_1 - \vec{v}_2) = \left\| \vec{v}_1 - \vec{v}_2 \right\| \tag{2}$$

Where v_1 and v_2 are RGB triplets (v = [R G B]).
The Vector Angle between two pixels is approximated by:

$$\sin\theta = \left(1 - \left(\frac{\vec{v}_1^{\,T} \vec{v}_2}{\left\| \vec{v}_1 \right\| . \left\| \vec{v}_2 \right\|} \right)^2 \right)^{1/2} \tag{3}$$

Because $\sin\theta \approx \theta$ for small angles. Again, v_1 and v_2 are RGB triplets (v = [R G B]). The Euclidean Distance and Vector Angle are combined using a saturation-based combination method. This combination is defined as:

$$C_{GV} = \rho(S_1,S_2)\sqrt{1-\left(\frac{\vec{v}_i^{\,T}(x,y)\vec{v}_0(x,y)}{\left\|\vec{v}_i(x,y)\right\|.\left\|\vec{v}_0(x,y)\right\|}\right)^2} + (1-\rho(S_1,S_2)).\left\|\vec{v}_i(x,y)-\vec{v}_0(x,y)\right\| \quad (4)$$

Where:

$$\rho(S_1,S_2) = \sqrt{\alpha(S_1).\alpha(S_2)} \quad (5)$$

And:

$$\alpha(S) = \frac{1}{1+e^{-slope(S-offset)}} \quad (6)$$

The "slope" and "offset" values in the sigmoid α(S) are set experimentally, and S_1 and S_2 are the saturation values of each pixel. This combination weights the Vector Angle more heavily in areas of high color saturation and the Euclidean Distance more heavily in areas of low saturation.

The algorithm for finding edges in the image is as follows:

1. For each pixel in the image, take the 3x3 window of pixels surrounding that pixel.

2. Calculate the saturation-based combination of the Euclidean Distance and Vector Angle between the center point and each of the eight points around it.

3. Assign the largest value obtained to the center pixel.

4. When each pixel has had a value assigned to it, run the results through a threshold to eliminate false edges.

E: Color Edge Detection using the Canny Operator

Another approach to edge detection using color information is simply to extend a traditional intensity based edge detector into the color space. This method seeks to take advantage of the known strengths of the traditional edge detector and tries to overcome its weaknesses by providing more information in the form of three color channels rather than a single intensity channel. As the Canny edge detector is the current standard for intensity based edge detection, it seemed logical to use this operator as the basis for color edge detection.

The algorithm we used for applying colors to the Canny edge detector was a very simple one:

1. Read in a color image and divide it into its three separate color channels.

2. Run each color channel through the Canny edge detector separately to find a resulting colored edge map.

3. Combine the resulting edge maps from each of the three color channels into one complete edge map. For this step there are a variety of ways you can combine the edges found for each different color, but we found that a simple additive approach provided the best results. So if there was an edge in any of the three colored edge maps, we added it to the general edge map.

F: Depth Edge Detection using Multi-Flash Imaging

This is another edge detector following the principle that using more data in the edge detection process should result in better detection of edges. However, in this case rather than merely extending from one channel of intensity to three channels of color, this edge detector actually makes use of multiple different images. The approach is based on taking successive photos of a scene, each with a different light source close to and around the camera's center of projection. The location of the shadows abutting depth discontinuities are used as a robust cue to create a depth edge map in both static and dynamic scenes. [5] The idea is that rather than using complicated mathematical techniques to try to extract edges from existing photographs, we should change the way we take photographs in general.

This technique uses a camera with four flashes located at cardinal directions around the lens to take four successive pictures. The differences in shadows between each picture suggest "depth edges", or edges caused by depth discontinuities in a scene. This method suppresses "texture edges", or edges caused by the texture of a surface which all lie at a relatively equivalent depth. This is accomplished by calculated the epipolar geometry of the *shadows* in the different images. The general algorithm is as follows:

1. Capture an image using only ambient light. Label this image as I_0.

2. For 'n' different light sources located a positions P_1-P_n, capture n pictures I^+_k, with k=1-n where I^+_k is the picture taken with light source position P_k

3. Remove the ambient component from each image: $I_k = I^+_k - I_0$

4. For all pixels x, $I_{max}(x) = max_k(I_k(x))$, k = 1..n. I_{max} is the base image, which is an approximation of what image you would get if the light source were exactly at the center of the camera lens.

5. For each image k, create a ratio image, R_k where $R_k(x) = I_k(x)/ I_{max} (x)$. The intensity of a point in an image, if it is lit, will follow the following equation:

$$I_k(x) = \mu_k \rho(x).(\hat{L}_k(x).N(x)) \tag{7}$$

where μ_k is the magnitude of the light intensity, p(x) is the reflectance at point X, $L_k(x)$ is the normalized light vector $L_k(x) = P_k - X$ and N(x) is the surface normal. If the surface is not lit, $I_k(x) = 0$. So the equation for the ratio is:

$$R_k(x) = \frac{I_k(x)}{I_{\max}(x)} = \frac{\mu_k(\hat{L}_k(x).N(x))}{\max_i(\mu_k(\hat{L}_i(x).N(x))} \tag{8}$$

However, if the objects are relatively diffuse and far from the camera relative to the positions of the light sources, the ratio can be approximated by simply $R_k(x) = \frac{\mu_k}{\max_i(\mu_i)}$ This ratio will be close to 1 in areas lit by light source k and close to 0 in areas not lit by light source k.

6. For each image R_k, traverse each epipolar ray from epipole e_k (location of the flashes). A sharp negative transition in the image indicates a depth edge. So if a pixel has a negative transition, mark that pixel as an edge. Since the flashes are oriented along the cardinal directions, tracing the epipolar rays is equivalent to walking along a row or column of the image.

(Algorithm taken from [5])

III. IMPLEMENTATION AND COMPARISON

All edge detectors were implemented using MATLAB. For the Marr-Hildreth edge detector, it was possible to set the slope threshold, the sigma of the Gaussian, and the size of the Gaussian. For the Canny edge detector and Color Canny edge detector, it was possible to set the high threshold and low threshold, the sigma for the Gaussian, and size of the Gaussian. For the Boolean Function Based edge detector, it was possible to set the local threshold and the global threshold. For the Euclidean Distance and Vector Angle Color edge detector, it was possible to set the slope and offset, and to set the final threshold. For the Multi-Flash edge detector, it was possible to set the threshold of the negative edge step.

A: Method for Comparison

There are five different criteria that are typically used for testing the quality of an edge detector:

-The probability of a false positive (marking something as an edge which isn't an edge)

-The probability of a false negative (failing to mark an edge which actually exists)

-The error in estimating the edge angle

-The mean square distance of the edge estimate from the true edge

-The algorithm's tolerance to distorted edges and features such as corners and junctions

(Criteria taken from [2])

However, in order to determine the third and fourth criteria, an exact map of the edges in an image must be known, and in general this is not available. It is also not plausible to assume that some "exact map" of all the edges can even be constructed. Therefore, the third and fourth criteria are not very useful. Additionally, corners and junctions simply are not handled well by any edge detector and must be considered separately. Therefore the fifth criterion is not very useful either.

The most important criteria are the first two, as it is much more important to have the proper features labeled as edges than having each edge exactly follow what would be considered the "ideal" edge or being able to handle special features such as corners and junctions. So for our evaluation, we only considered the first two criteria.

IV. EXPERIMENTAL RESULTS

We ran five separate test images through our edge detectors (except the Multi-Flash edge detector, which only had data sufficient data to run on two of the images). One image was artificial and the rest were real world photographs. The results are shown in the figures below. All color images were converted to grayscale using *Matlab's* RBG2GRAY function except when using an edge detector that required color information. Various threshold, sigma, slope, etc. values were chosen by hand. The images were blurred by a Gaussian filter (3x3 filter, sigma=1) before being fed into the Euclidean Distance and Vector Angle color edge detector as we found this significantly decreased the effect of noise. The other edge detectors do their own smoothing (Where applicable). The low threshold in the Canny edge detector was always defined to be 40% of the high threshold. The average of the four images from the Multi-Flash edge detector was used as input to the other edge detectors.

The first thing to notice about the Boolean and Euclidean Distance/Vector Angle edge detectors is that neither algorithm identifies edges down to a single point as in Canny's edge detector. The edges are often spotty and disconnected. Edge following to trace out a continuous edge is not possible in these edge detectors because the direction of the edge is not known (since they are not based on the gradient of the image). The Marr-

Hildreth edge detector will give more nicely connected edges if hysteresis is used to threshold the image, and will not give connected edges if a single threshold is used. Regardless, it is usually gives spotty and thick edges.

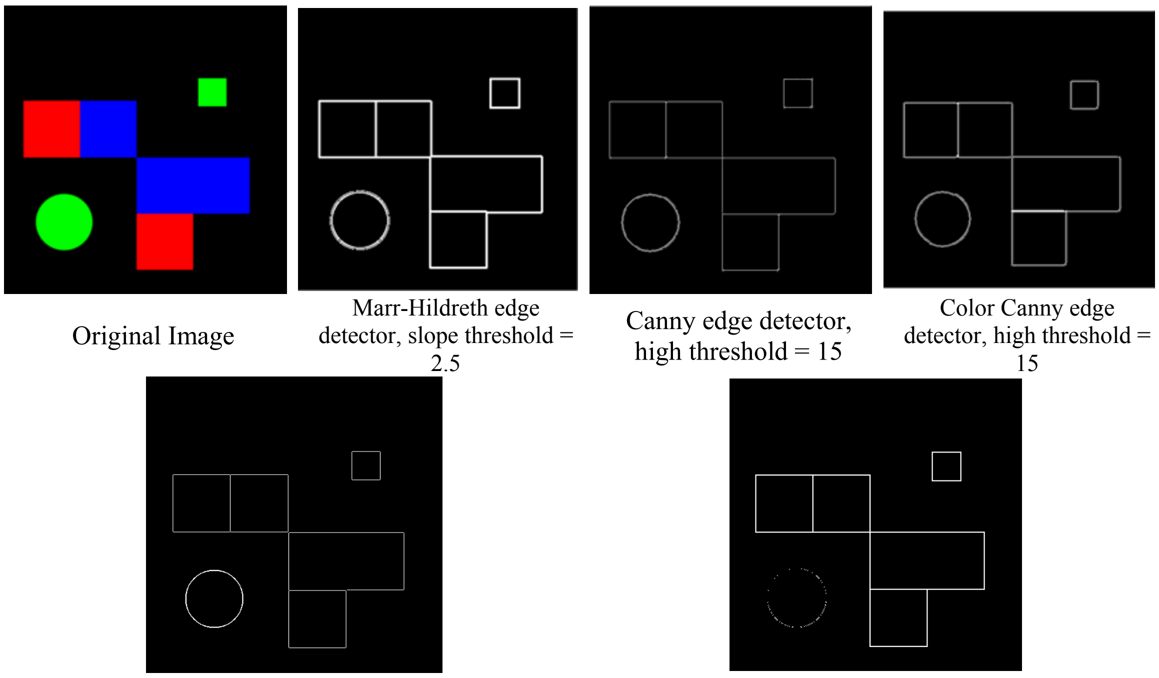
Figure 1 shows the ability of the edge detectors to handle corners as well as a wide range of slopes in edge on the circle. The Canny edge detector becomes fairly confused at corners due to the Gaussian smoothing of the image. Also, since the direction of the edge changes instantly, corner pixels looks in the wrong directions for its neighbors. The color version of Canny produces many double lines as the edge may be detected in a different location in each channel. The Boolean edge detector usually omits the corner pixel since the pattern for a corner is not one of the bit masks. Other than the omitted pixels, the Boolean edge detector performs well on the boxes and circle. The Euclidean Distance/Vector angle edge detector performs well on the blocks, creating a 2 pixel wide line along every edge. It became a little confused on the circle. This could possibly be remedied with a different threshold between the Euclidean Distance metric and the Vector Angle metric. The Marr-Hildreth edge detector creates lines even thicker than the Euclidean Distance.

Figure 2 is the standard edge detector benchmarking image. Overall, the Boolean edge detector performs a decent job of marking the locations of edges, but many of the edges are spotty and not contiguous. For the most part it detects the same edges as the *Canny* edge detector. The color version of the *Canny* detector was able to find a few more edges than the grayscale version given the same input parameters. These edges are most noticeable on the hat. The Euclidean Distance detector creates much wider edges than the other two methods. It appears as if it marks edge pixels on both sides of the edge. As a result the output image shows the major edges, but not much fine grained detail. In general, the Boolean edge detector makes no guarantees about finding thin edges, but it usually does a reasonable job.

Figure 3 is a picture of a shoreline. All edge detectors had problems detecting the different ridges of the cliff. The foam of the waves also provided some inconsistent results. There are a lot of discrepancies in color at these locations, but no clear edges. Similar to the Figure 2, the Euclidean Distance detector produces much thicker lines and less detail than the other edge detectors. The Boolean edge detector does a better job of maintaining contiguous curves for the edges, but they still have a few discontinuities.

Figure 4 is the first of the Multi-Flash images [5]. The bright flashes in the different directions caused drop shadows around some edges of the object. Even though the average of all four images was used as the input to the other edge detectors, some of them still picked up on the faint shadows. Surprisingly, the Multi-Flash edge detector did not perform better than other edge detectors for this image. For example, it missed parts of the vertebrae. It could pick up these images if the threshold was reduced, but much more noise was introduced into the image. Other than the Boolean edge detector, the Multi-Flash method is the only detector that no smoothing of the image before processing, so it is more sensitive to noise in the image. The other edge detectors all identify almost the same edges, but follow similar behavior to previous images.

Figure 5 more accurately shows the capabilities of each edge detector. The Marr-Hildreth detector perceives many edges, but they are too spotty and wide to really identify any features. The Canny edge detector gives nice outlines of the table, the vase, and many of the flowers on the border. Features in the middle of the arrangement are missed, but some are recovered with the addition of color. For example, the red flower in the center and the leaves to its right are found. The Boolean edge detector does a good job at detecting a large number of edges, but many of them are noisy. The Euclidean Distance/Vector Angle edge detector finds strong edges around the color flowers, but finds almost no edges in the middle of the green leafy region. It also misses the table entirely. The Multi-flash edge detector finds most of the geometry in the scene but misses parts of the table. If the background is too far away from the objects, the drop shadows will not be sharp enough to detect a discontinuity. The inventors of this algorithm supplement the edge detection with a step that finds the color difference between the object and its background.


Original Image

Marr-Hildreth edge detector, slope threshold = 2.5

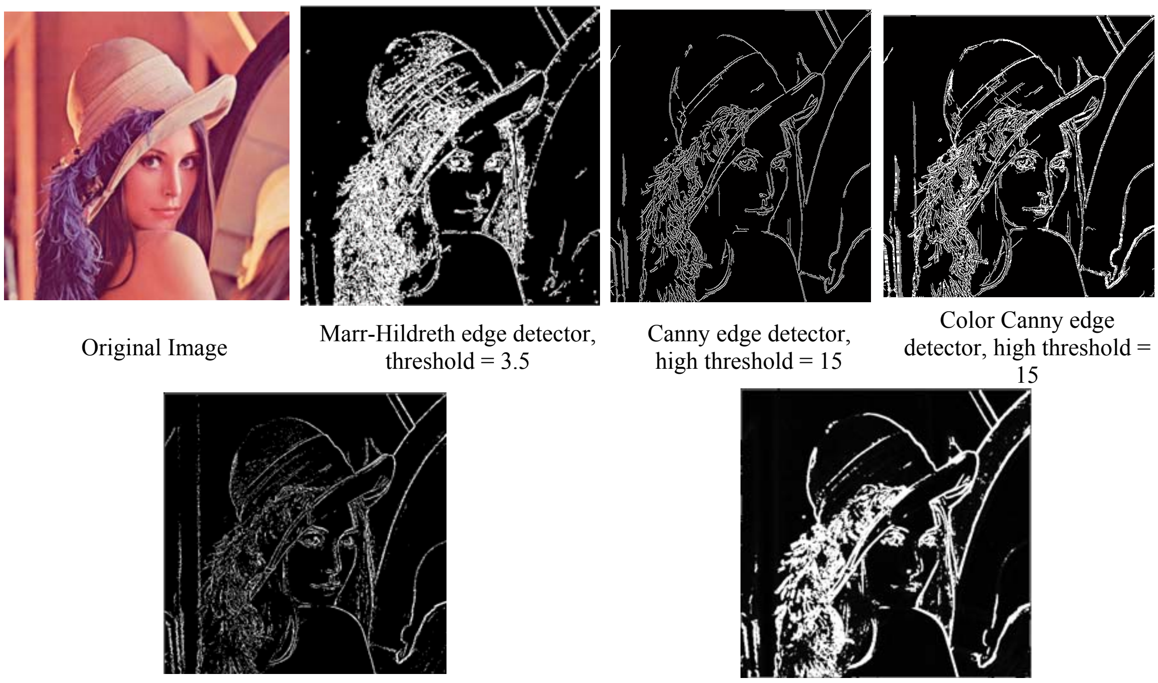
Canny edge detector, high threshold = 15

Color Canny edge detector, high threshold = 15

Boolean edge detector, Tn = 40, C = 4

Euclidian Distance/Vector Angle Detector

Figure 1



Original Image

Marr-Hildreth edge detector, threshold = 3.5

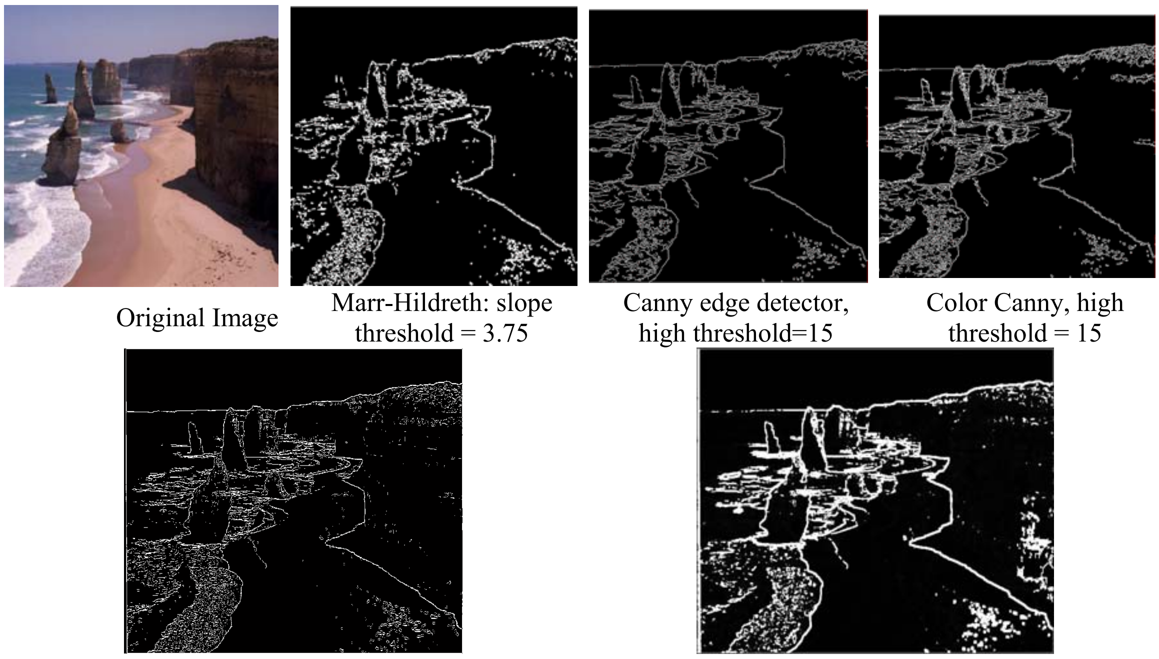
Canny edge detector, high threshold = 15

Color Canny edge detector, high threshold = 15

Boolean edge detector, Tn = 60, C = 0

Euclidian Distance/Vector Angle Detector

Figure.2



Original Image

Marr-Hildreth: slope threshold = 3.75

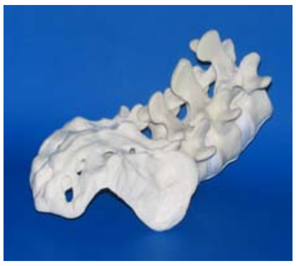
Canny edge detector, high threshold=15

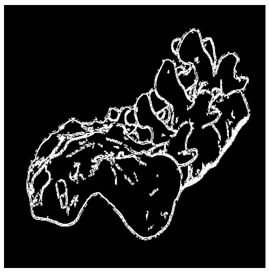
Color Canny, high threshold = 15

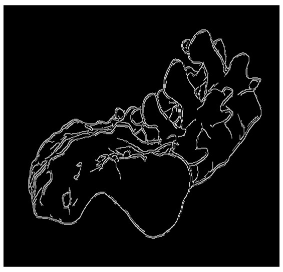
Boolean edge detector, Tn = 70, C = 1

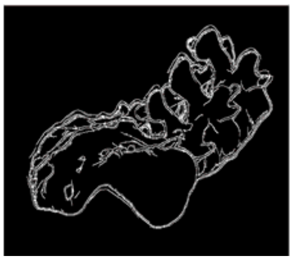
Euclidean Distance/Vector Angle Detector

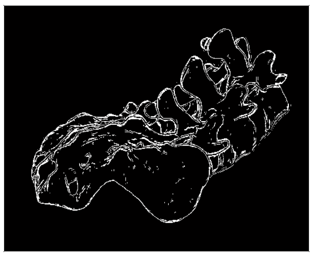
Figure.3

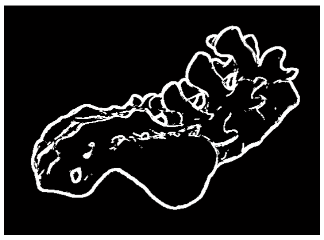

Original Image (upper flash shown)


Marr-Hildreth: high slope threshold = 3


Canny edge detector, high threshold = 15


Color Canny edge detector, high threshold = 15


Boolean edge detector Tn = 50, C = 4


Euclidian Distance/Vector Angle Detector

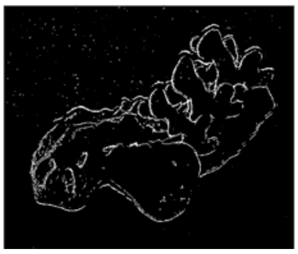
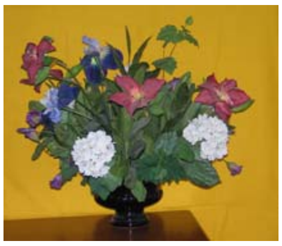
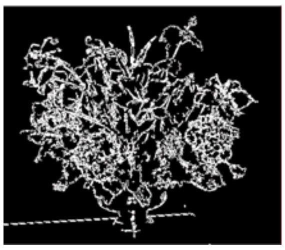
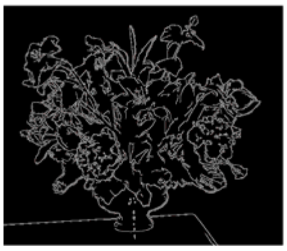
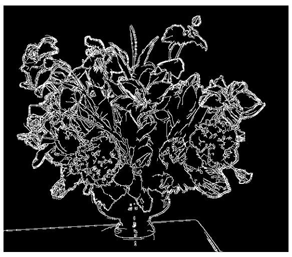

Multi-flash edge detector: Threshold = 0.8

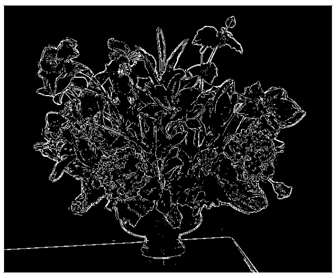
Figure.4

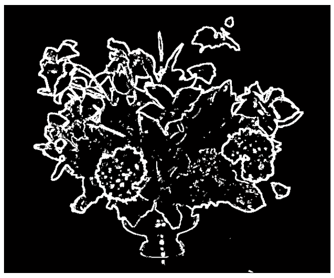

Original Image (upper flash shown)


Marr-Hildreth edge detector with slope threshold = 4.5


Canny edge detector, high threshold = 15


Color Canny edge detector, high threshold = 15


Boolean edge detector Tn = 70, C = 2


Euclidian Distance/Vector Angle edge detector

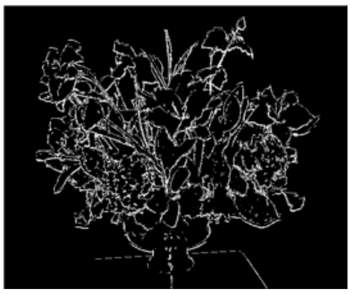

Multi-flash edge detector with threshold = 0.9

Figure.5

V. CONCLUSION

The Boolean edge detector performs surprisingly similarly to the Canny edge detector even though they both take drastically different approaches. Canny's method is still preferred since it produces single pixel thick, continuous edges. The Boolean edge detector's edges are often spotty. Color edge detection seems like it should be able to outperform grayscale edge detectors since it has more information about the image. In the case of the Canny color edge detector, it usually finds more edges than the grayscale version. Finding the optimal way to combine the three color challenges may improve this method. The other color edge detection scheme we implemented, the Euclidian Distance/Vector Angle detector, did a decent job of identifying the borders between regions, but misses fine grained detail. If the direction of the edge were known, a non-maximal suppression on the color edge detector's output may help the granularity of its output. Multi-flash edge detection shows some promise as it strives to produce photographs that will be easy to edge detect, rather than running on an arbitrary image. One problem inherent to the Multi-flash edge detector is that it will have difficulty finding edges between objects that are at almost the same depth or are at depths which are very far away. For example, the Multi-flash method would not work at all on an outdoor scene such as the shoreline.

REFERENCES

[1] M.B. Ahmad and T.S. Choi , Local Threshold and Boolean Function Based Edge Detection, *IEEE Transactions on Consumer Electronics*, *Vol. 45, No 3. August 1999*

[2] R. Owens, "Lecture 6", Computer Vision IT412, 10/29/1997. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT6/node2.html

[3] S. Price, "Edges: The Canny Edge Detector", July 4, 1996. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/canny.htm

[4] S. Wesolkowski and E. Jernigan, "Color Edge Detection in RGB Using Jointly Euclidean Distance and Vector Angle", *Vision Interface '99*, Trois-Rivieres, Canada, 19-21 May

[5] R, Raskar; Tan, K-H; Feris, R.; Yu, J.; Turk, M., "Non-photorealistic Camera: Depth Edge Detection and Stylized Rendering Using Multi-Flash Imaging", *ACM SIGGRAPH*, August 2004

[6] Jean Ponce, “Lecture 26: Edge Detection II”, 12/2/2004. http://www-cvr.ai.uiuc.edu/~ponce/fall04/lect26.ppt

Received: December 31, 2007