ASB: A Fast, Robust Edge Detection Algorithm.

Aashutosh Singh Baghel.
Department of Computer Science and Engineering
Samrat Ashok Technological Institute
Vidisha, Madhya Pradesh, India
aashubaghel08@gmail.com

Som Singh Thakur
Department of Computer Science and Engineering
Samrat Ashok Technological Institute
Vidisha, Madhya Pradesh, India
tsom179@gmail.com

ABSTRACT - This paper presents a custom algorithm for edge detection in digital images through pixel-level analysis and proximity-based difference computation. The method involves converting RGB image data into numerical representations (K values), calculating the difference (V) between a pixel and its neighbours, and applying a filtering mechanism based on a threshold to detect edges. A corrective mechanism is proposed to handle image artefacts such as lens glares and light smudges using Gaussian blur. While the approach can yield accurate contour maps, it suffers from high computational demands due to per-pixel processing. The study evaluates the effectiveness of the method against standard edge detection algorithms such as Sobel and Canny, and proposes enhancements such as pre-filtering and block-level processing to improve performance. Experimental results on 512x512 PNG images demonstrate the strengths and limitations of the approach, offering insights into optimizing edge detection under varied imaging conditions.

I. Introduction

An image is a visual representation. It can be two-dimensional, such as a drawing or painting, or a photograph, or three-dimensional, such as a carving or sculpture. Images may be displayed through other media, including a projection on a surface, activation of electronic signals, or digital displays.
A digital image comprises of a grid of pixels, where each pixel has 3 channels of data that hold colour values, while one channel of data that holds the transparency of each pixel. The three colour channels are the RGB or the CMY channels [1].
In images, *edges* are the sharp changes in colour, brightness or the saturation of pixels. Edges generally occur at the boundaries of objects within the images. These edges can be generated due to a colour change in the objects in the image, difference in object distance or depth from the camera. An edge detection model does the job of identifying these edges within an image. It needs to be robust to counter noise within the image, and be fast to be used for real-time applications. In this study we have aimed at introducing a new edge detection algorithm: ASB. It works by calculating the nearby pixel differences and applying a filtering mechanism based on the amount of differences encountered in nearby pixels.

II. Pre-processing of image

In an image, unwanted details like environmental noise, dust and film grain cause false and weak edges. These are not *true* to the real nature of the image and need to be removed or corrected for.
To combat environmental noise and grain in the image, other algorithms generally use Gaussian blurring to reduce the amount of noise in the image[2][3]. Although it is effective in reducing fake edges it adds an extra step in the model. Under ASB, there are two modes: either using a Gaussian blur to reduce noise or increasing the area of calculating pixel difference. The advantages and disadvantages of both approaches are discussed further on.
For now, every pixel in the image needs to be converted to operable units. The image is loaded in memory and every pixel is interpreted and stored as an integer.
A 3d matrix is generated of the size [L×M×3]

- ‘L’ and ‘M’ correspond to the horizontal and vertical size of the image.

- The data of the individual channels per pixel can be accessed by altering the third variable in the matrix index.

Here the particular channel value of a particular pixel can be represented as:

$$K_{i,j,c} = (C)$$

- 'i' and 'j' are the position of the pixel in the image, while c is the colour channel and C is the value of the element at (i,j,c).

The value of K can be between 0 and 3*2^n where n is the bit depth of the image. Images with higher bit depth require more memory as the integer value of K uses more memory.

III. Near Pixel differences.

For a particular pixel, the difference between its **K** value and the **K** value of it's nearby pixels is calculated. Similarly to how **K** is stored, the sum of the differences between all nearby pixels is stored for every pixel in a separate matrix. This second matrix is called a variance grid, where every element stores the variance of pixel values for each corresponding pixel. Let us call these sums of differences or variance values as **V**.

$$\boldsymbol{V_{i,j} = \sum_{c=0}^{2} \sum_{m=l-1}^{l+1} \sum_{l=i-1}^{i+1} \left[K_{i,j,c} - K_{l,m,c}\right]}$$

IV. Filtration and Correction of boundaries

Before proceeding further, we need to address the pixels which lie on the boundary of the image. While calculation of the variance matrix, pixels on the boundaries aren't truly accounted for. Due to the nature of the algorithm, the variance (**V**) of the pixels on the boundaries is sure to be less than others. This is because they lack nearby pixels to calculate difference from. This can be corrected by multiplying the pixels on the boundaries by a value that increases the **V** value of the pixel to what it should have been.

Let us call this factor '**β'.** If **Z** is the number of *missing* pixels, formula for **β** will be:

$$\boldsymbol{\beta = \frac{Z\left|\sum_{x=1}^{L-1}\sum_{y=1}^{M-1} V_{x,y}\right|}{LM}}$$

After the corrected variance matrix has been calculated and boundaries have been accounted for, a threshold can be applied to filter the edges from the non edges. This threshold value needs to be such that it accounts for most of the edges in the image while keeping the amount of non-edges to a minimum. A good measure of such threshold is the root mean square of the variance values. Let us call the threshold '**α**'. The formula for **α** will be

$$\boldsymbol{\alpha = \sqrt{\frac{1}{LM}\sum_{i=1}^{L}\sum_{j=1}^{M} V_{ij}^2}}$$

Here:

1. 'L' is the vertical size of the image.
2. 'M' is the horizontal size of the image.

3. V is the variance per pixel.

The RMS of the variance offers the best quality result while also eliminating most non-edges.
After threshold has been calculated, the final image can be generated. For every pixel whose **V** exceeds **α**, the pixel can be represented as white, otherwise as black. The final image will have white pixels over all the edges that the algorithm seems to think is an edge.

V. Results

The images used are PNG encoded. No transparency is present in any of the photo.

Figure 1: [Top to bottom] Original image, variance map represented as an image, Final image

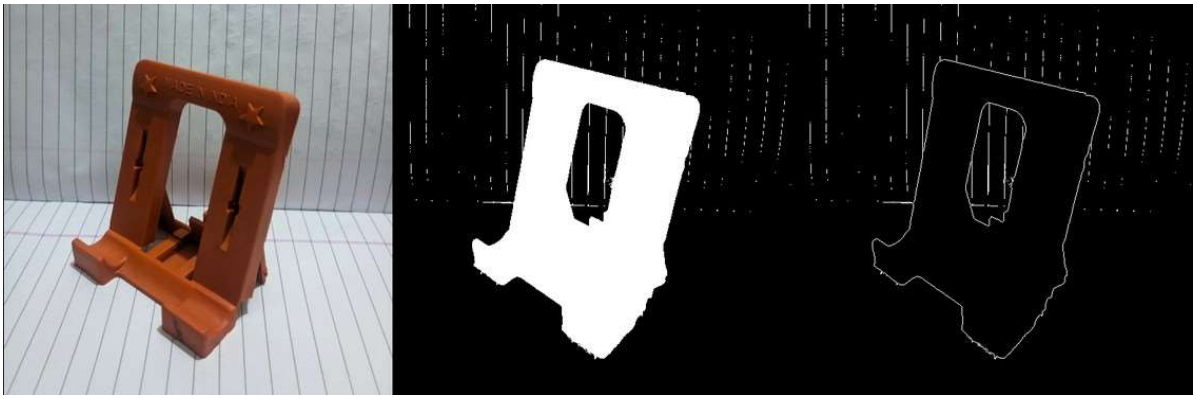

Figure 2: [Left to right] Original image, variance map represented as an image, Final image

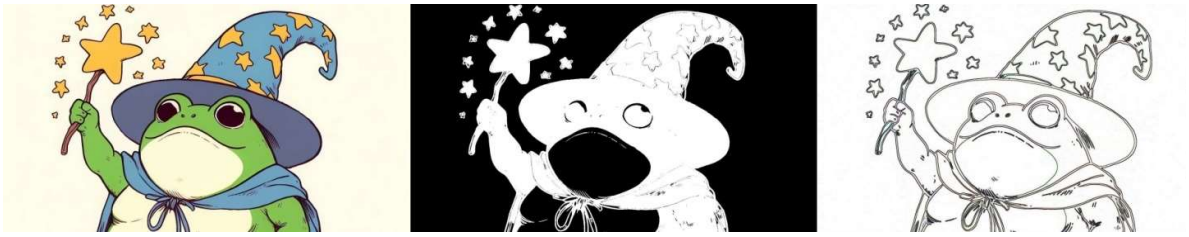

Figure 3: [Left to right] Original image, variance map represented as an image, Final image

VI. Downsides

1. High computational requirement: The algorithm does not focus on the compressed blocks of image but rather on individual pixels. Due to this inherent flaw, the computation power required is high (comparable to Canny)[4]. This limits usage in video feeds which require just-in-time conversion. Ideally the image block should be processed without full decoding. This however would pose the problem of re-writing the program for every codec.
   Although the algorithm is already very fast in the state it is presented here, codec-centric re-writing and offloading matrix calculation to the GPU can offer significantly decrease processing times.

2. High bitrate requirement: Increasing the number of nearby pixels to calculate **V**[see II], increases the computational load by factor of block size. On low bitrate images, this offers diminishing returns as compared to other prevalent methods for edge detection. As a result, the model works faster on higher bitrate image while it takes more compute to process low bitrate images.
   NOTE: The output images are still very good quality even for low quality images with only adjacent pixels used for variance calculation.

3. Fake edges: Due to low bitrate images, fake contours are detected. These appear as a grid over the entire image. This is due to the 'block size' that the image codec uses for compression. The boundaries of these blocks sometimes present themselves as edges. This can also be combated by:
   a. Gaussian blurring the whole image
   b. Applying Gaussian blur locally to block boundaries
   c. Using a high-bitrate image
   d. Increasing block size

4. Soft edges and discontinued edges: In images where objects have been blended together due to external factors such as an unstable camera, glares, movement of object and relatively slow shutter speeds, some edges get *dissolved*. This poses a problem as details are lost within the smudges. This can be combated by either
    a. Increasing the count of pixels used for variance matrix (**V)** calculation
    b. Applying a sharpening filter such as High-Boost or Notch filters.

VII. Acknowledgements

I would like to express my sincere gratitude to Dr. Kanak Saxena, Head of the Department of Computer Science and Engineering, for their constant support, guidance, and encouragement throughout the course of this research. Their valuable insights and constructive feedback have been instrumental in shaping this project.
I am also deeply thankful to Asst. Prof. Mukesh Azad for providing post-haste clarification. I am also deeply thankful to the Institute of Management Studies (IMS), Ghaziabad Committee for providing me with the opportunity to present my research at a reputable platform. Without the support of any of the previously mentioned parties, this research would not have been possible.

VII. References

[1] "Meaning and Function of a Picture, published by Taylor & Francis, Ltd. On behalf of the Mathematical Association of America, DOI: 10.2307/2301228on Jstor.Org". JSTOR 2301228
[2] Dim, Jules R.; Takamura, Tamio (2013-12-11). "Alternative Approach for Satellite Cloud Classification: Edge Gradient Application"
[3] https://www.projectrhea.org/rhea/index.php/Edge_Detection_with_Gaussian_Blur by Buyue Zhang
[4] Martin KRÁLIK, Libor LADÁNY “Canny Edge Detector Algorithm Optimization Using 2D Spatial Seperable Convolution”
[5] https://www.mathworks.com/help/images/block-processing-large-images.html