# REPORT OF AN EDGE DETECTION PROGRAM USING TAYLOR SERIES

Written By

Kreshnayogi Dava Berliansyach 22/496686/PA/21352

**DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS**
**FACULTY OF MATHEMATICS AND NATURAL SCIENCE**
**UNIVERSITAS GADJAH MADA**
**YOGYAKARTA**
**2024**

## INTRODUCTION

In this report, we are going to discuss a program that is able to obtain the edges of an object in an image. The aim of this report is to understand how this program works and evaluate its effectiveness in finding object edges accurately and efficiently. One specific program will be focused on and its performance will be tested. The weaknesses of the program will also be discussed.

## THE PROGRAM

The program itself has three parts, two functions and one main function. The first function is used to change the image to a grayscale image. This is done so that it is easier for the program to process it and acquire its edges. The second function is used to get the edges of an object in an image that is already grayscale. The main function is used to run the codes.

### A. THE LIBRARIES

The libraries used in this program are as follows:

```
import cv2
import numpy as np
from google.colab import drive
import matplotlib.pyplot as plt
```

### B. RGB TO GRAYSCALE FUNCTION

The code:

```
def rgb_to_gray(rgb_image):

    # Get image dimensions
    height, width, _ = rgb_image.shape

    # Initialize grayscale image array
    grayscale_image = np.zeros((height, width), dtype=np.uint8)

    # Iterate over each pixel
    for y in range(height):
        for x in range(width):
```

```
        # Extract RGB values
        red = rgb_image[y, x, 0]
        green = rgb_image[y, x, 1]
        blue = rgb_image[y, x, 2]

        # Apply the NTSC formula for grayscale conversion
        grayscale_value = int(0.2989 * red + 0.5870 * green + 0.1140 * blue)

        # Assign grayscale value to the corresponding pixel
        grayscale_image[y, x] = grayscale_value

    return grayscale_image
```

The function above takes an rgb image as an input and returns a grayscale version of the image. It begins by obtaining the height and width of the image received and initializes an empty array called grayscale_image to store the resulting grayscale image. It then goes over each pixel in the image, extracting the red, green, and blue values. It extracts the color by using the NTSC formula. The NTSC formula represents the percentage of red, blue, and green colors that a person can perceive, expressed as a formula. The formula is then used to calculate the corresponding grayscale value for each pixel and put that value to the corresponding position in the grayscale image array. This process repeats until the last pixel where then the function will return the grayscale image. Important to note that 'dtype=np.uint8' at the array initialization line is used to give the array an 8-bit integer data type.

## C. EDGE DETECTION FUNCTION

The code:

```
def edge_detection(gray_image):
    threshold = 20

    # Get image dimensions
    height, width = gray_image.shape

    # Initialize grayscale image array
    edges = np.zeros((height, width), dtype=np.uint8)

    # Iterate over each pixel
    for y in range(1, height - 1):
        for x in range(1, width - 1):
            # Compute first-order derivatives using Taylor series expansion
            dy = (int(gray_image[y + 1, x]) - int(gray_image[y - 1, x]))/2
            dx = (int(gray_image[y, x + 1]) - int(gray_image[y, x - 1]))/2

            # Magnitude calculation
            magnitude = np.sqrt(dx*dx + dy*dy)

            # Apply threshold
            if magnitude > threshold:
                edges[y, x] = 255
            else:
                edges[y, x] = 0

    return edges
```

The function above obtains an input in the form of a grayscale image. It begins by setting a threshold value to determine the significance of a pixel which will represent the edges of an object in the grayscale image. It then obtains the height and width of the image received and initializes an empty array called edges to store the resulting edges of the grayscale image. It will then iterate over each pixel of the grayscale image, excluding the border pixels, this is to ensure that the derivative calculations consider neighboring pixels properly. The function then calculates the first-order derivatives using the central difference Taylor Series. The derivative values, dx and dy, are then used to compute the magnitude of the gradient at each pixel using the formula in the magnitude line. The magnitude would then be compared to the threshold to determine the significance of the pixel. If the pixel is significant to the edges of the object, the pixel would then be given the value of 255 which is the color white, otherwise the pixel would be given the value of 0 which is the color black. After iterating over all pixels, the function returns the edges array. Important to note is the 'int' usage in the taylor series formula of dx and dy. The int important is because the values in the array gray_image is still in integer 8 bit data type. The 8-bit integer data type can have the value between 0 to 255 and only that, so when you get a negative value, which there is a possibility of in the case of the taylor series formula, the value would instead be wrapped around and back to the top which would be bad because if we get the value of -5, the value would instead be 251.

## D. MAIN FUNCTION

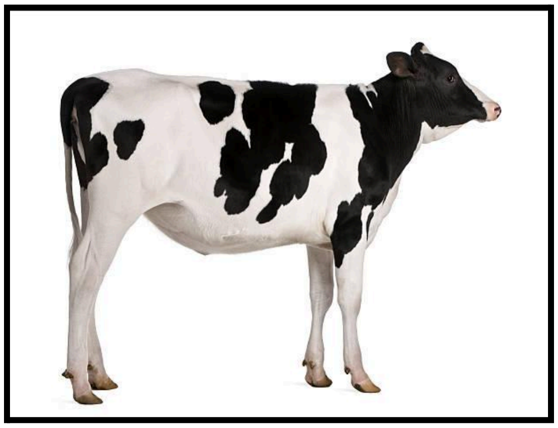
The code:

```
# Mount Google Drive
drive.mount('/content/drive')

# Path to the image file on Google Drive
rgb_image_path = '/content/drive/My Drive/cow.jpg'

# Read the RGB image
rgb_image = cv2.imread(rgb_image_path)

gray_image = rgb_to_gray(rgb_image)

# Apply edge detection using Taylor series approximation
edges = edge_detection(gray_image)

# Display the edging
plt.imshow(edges, cmap='gray')
plt.axis('off')  # Hide axes
plt.show()
```

Since the code was run on google colab, the library ‘drive’ is used to input an image from google drive. The main function starts off with mounting google drive onto the google colab environment then saving an image path to ‘rgb_image_path’. In this case, a file called cow.jpg will be used. It then uses OpenCV's cv2.imread() function to read the image path and retrieve the image file. Then the function rgb_to gray is ran on the image retrieved and the result is saved to ‘gray_image’ where it would then be inputted to the function edge_detection and its result would be saved on to ‘edges’. Finally, Matplotlib is employed to display the resulting edges of the image. The edges are visualized as a grayscale image, and the axes are hidden for a cleaner presentation.

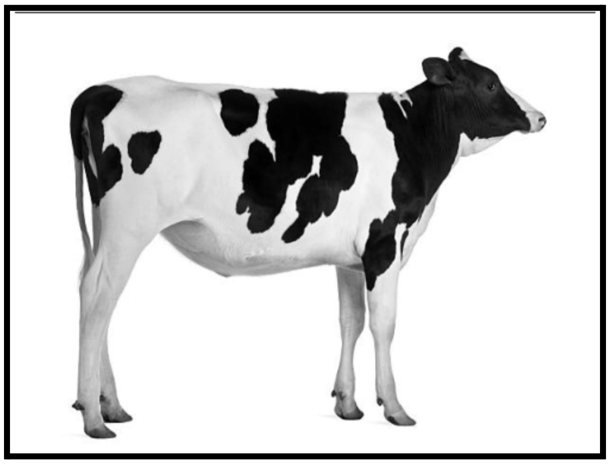
## THE RESULT

### A. ORIGINAL IMAGE



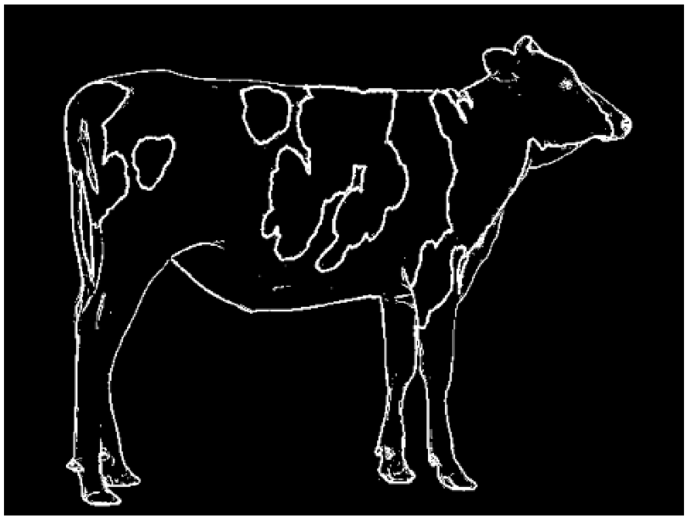
This picture of a cow will be used on the program. It has a pixel size of 612x 464.
The link to the picture can be found here: cow.jpg

### B. GRAYSCALE IMAGE



The image above is the result of the image after going through the grayscale function.

## C. EDGES IMAGE



The image above is the result of the edges retrieved from the grayscale image.

## CONCLUSION

In conclusion, the utilization of Taylor series in determining the edges of an object presents a viable and efficient approach to image processing and edge detection. Through the application of mathematics, particularly the Taylor series, the program is able to accurately identify the boundaries of objects.