To study Edge detection with Canny
Date of Performance: 17/07/2023
Date of Submission: 24/07/2023

Aim: To study Edge detection with Canny

Objective:Perform Canny Edge detector using Noise reduction using Gaussian filter ,Gradient calculation along the horizontal and vertical axis Non-Maximum suppression of false edges, Double thresholding for segregating strong and weak edges ,Edge tracking by hysteresis

Theory: The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

What are the three stages of the Canny edge detector To fulfill these objectives, the edge detection process included the following stages.

- Stage One - Image Smoothing.
- Stage Two - Differentiation.
- Stage Three - Non-maximum Suppression.

The basic steps involved in this algorithm are:

● Noise reduction using Gaussian filter

● Gradient calculation along the horizontal and vertical axis

● Non-Maximum suppression of false edges

● Double thresholding for segregating strong and weak edges

● Edge tracking by hysteresis

Now let us understand these concepts in detail:

1. Noise reduction using Gaussian filter:
This step is of utmost importance in the Canny edge detection. It uses a Gaussian filter for the removal of noise from the image, it is because this noise can be assumed as edges due to sudden intensity change by the edge detector. The sum of the elements in the Gaussian kernel is 1, so the kernel should be normalized before applying convolution to the image. In this Experiment, we will use a kernel of size 5 X 5 and sigma = 1.4, which will blur the image and remove the noise from it. The equation for Gaussian filter kernel is

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

after applying these kernel we can use the gradient magnitudes and the angle to further process this step. The magnitude and angle can be calculated as

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x, y) = arctan\left(\frac{I_y}{I_x}\right)$$

2. Non-Maximum Suppression:

This step aims at reducing the duplicate merging pixels along the edges to make them uneven. For each pixel find two neighbors in the positive and negative gradient directions, supposing that each neighbor occupies the angle of pi /4, and 0 is the direction straight to the right. If the magnitude of the current pixel is greater than the magnitude of the neighbors, nothing changes, otherwise, the magnitude of the current pixel is set to zero.

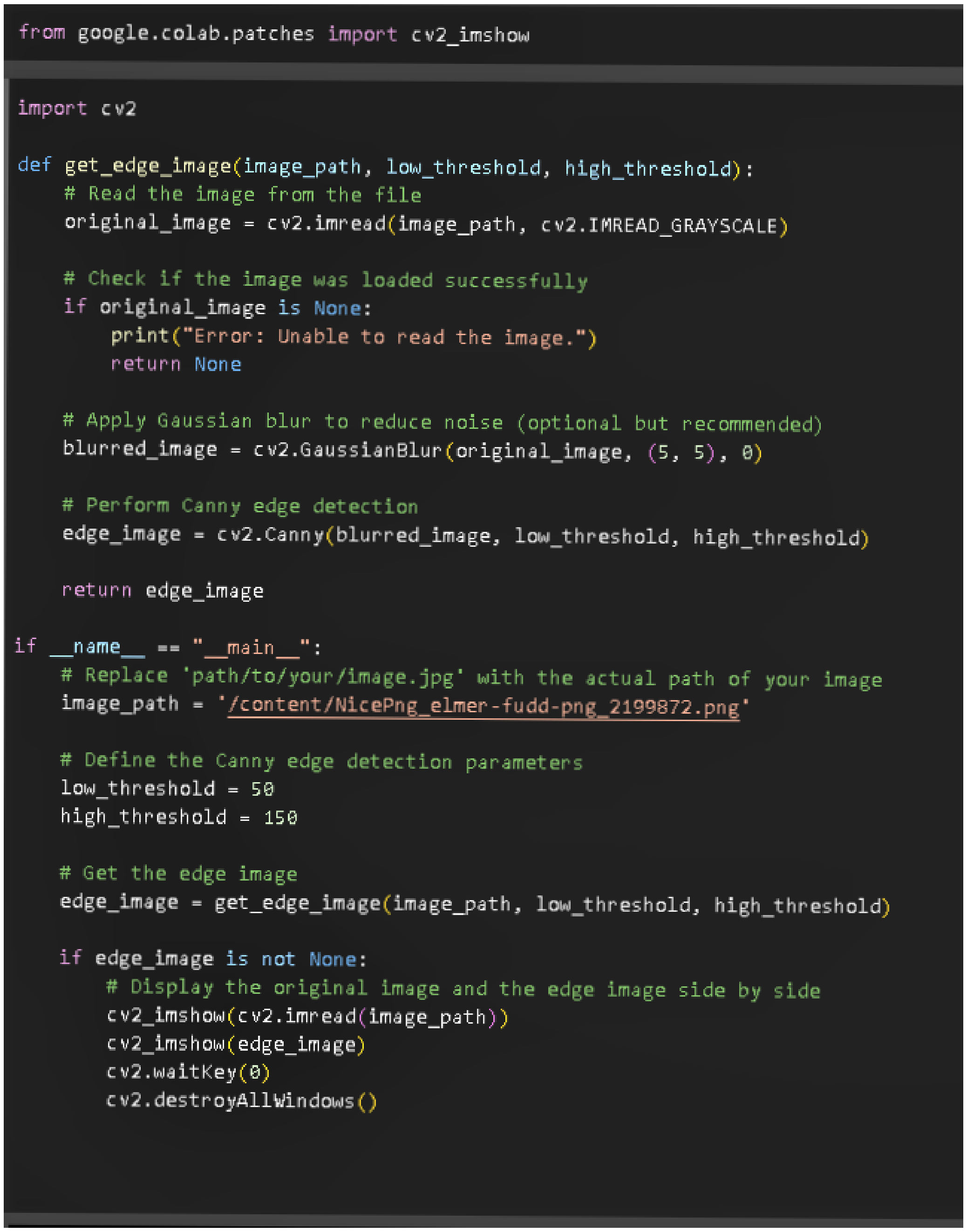
3. Double Thresholding

The gradient magnitudes are compared with two specified threshold values, the first one is lower than the second. The gradients that are smaller than the low threshold value are suppressed, the gradients higher than the high threshold value are marked as strong ones and the corresponding pixels are included in the final edge map. All the rest gradients are marked as weak ones and pixels corresponding to these gradients are considered in the next step.

5. Edge Tracking using Hysteresis

Since a weak edge pixel caused by true edges will be connected to a strong edge pixel, pixel W with weak gradient is marked as edge and included in the final edge map if and only if it is involved in the same connected component as some pixel S with strong gradient. In other words, there should be a chain of neighbor weak pixels connecting W and S.After that, you can decide which pixels will be included in the final edge map.

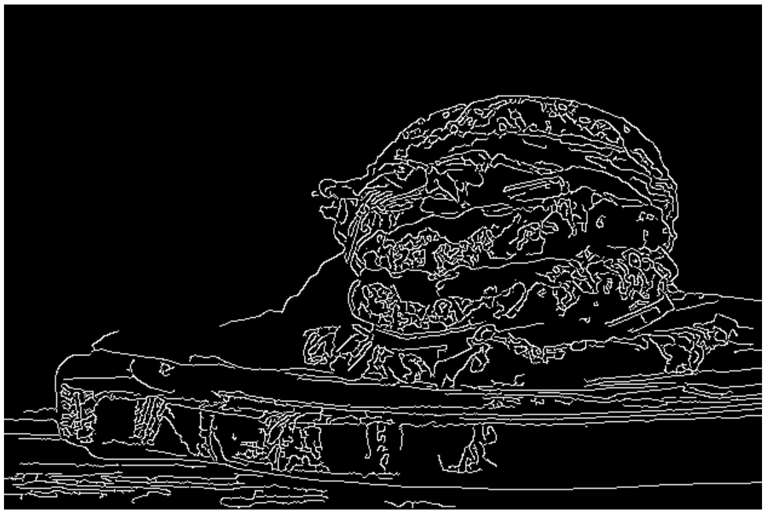
Below is the implementation.:



```
from google.colab.patches import cv2_imshow
```

```
import cv2

def get_edge_image(image_path, low_threshold, high_threshold):
    # Read the image from the file
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded successfully
    if original_image is None:
        print("Error: Unable to read the image.")
        return None

    # Apply Gaussian blur to reduce noise (optional but recommended)
    blurred_image = cv2.GaussianBlur(original_image, (5, 5), 0)

    # Perform Canny edge detection
    edge_image = cv2.Canny(blurred_image, low_threshold, high_threshold)

    return edge_image

if __name__ == "__main__":
    # Replace 'path/to/your/image.jpg' with the actual path of your image
    image_path = '/content/NicePng_elmer-fudd-png_2199872.png'

    # Define the Canny edge detection parameters
    low_threshold = 50
    high_threshold = 150

    # Get the edge image
    edge_image = get_edge_image(image_path, low_threshold, high_threshold)

    if edge_image is not None:
        # Display the original image and the edge image side by side
        cv2_imshow(cv2.imread(image_path))
        cv2_imshow(edge_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

Output:



Conclusion: The Canny Edge detector uses smart steps to find edges in images. It starts by reducing noise with a special filter. Then, it finds how fast the brightness changes using horizontal and vertical lines. It's careful to remove fake edges. Next, it separates strong and weak edges using two levels. Lastly, it follows edges to make sure it's finding the right ones. This all adds up to really good edge detection.