DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

Assignment#2

DIGITAL IMAGE PROCESSING

Student Name: _____Salva Nadeem_____

Degree/Syndicate: 43 CE B

Abstract

This code implements an algorithm to analyse fundus images to locate the optic disc and assess its characteristics.The process involves several steps:

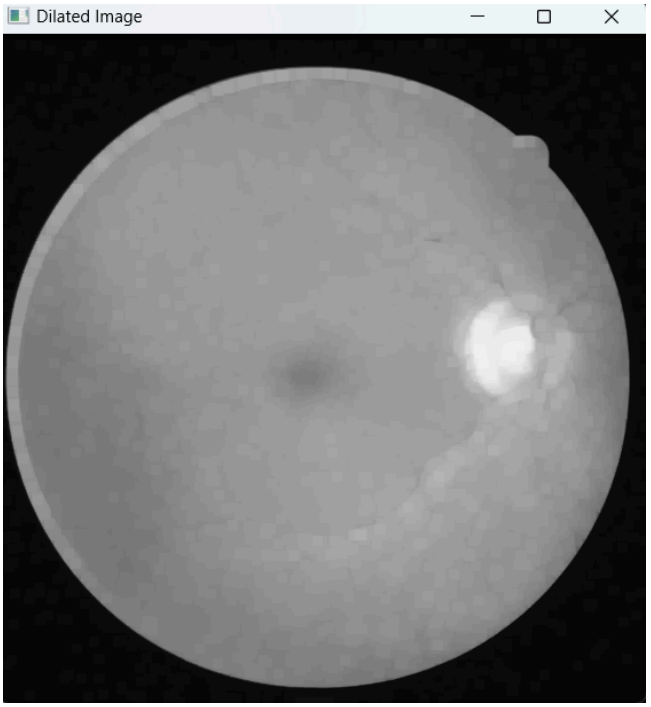
Preprocessing: The fundus images are preprocessed to enhance vein structures using morphological operations and adaptive thresholding.

The Original Image:

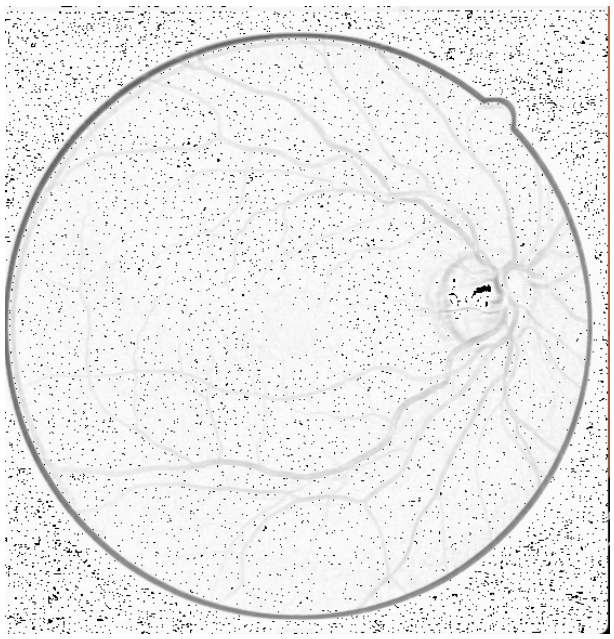
Functions:

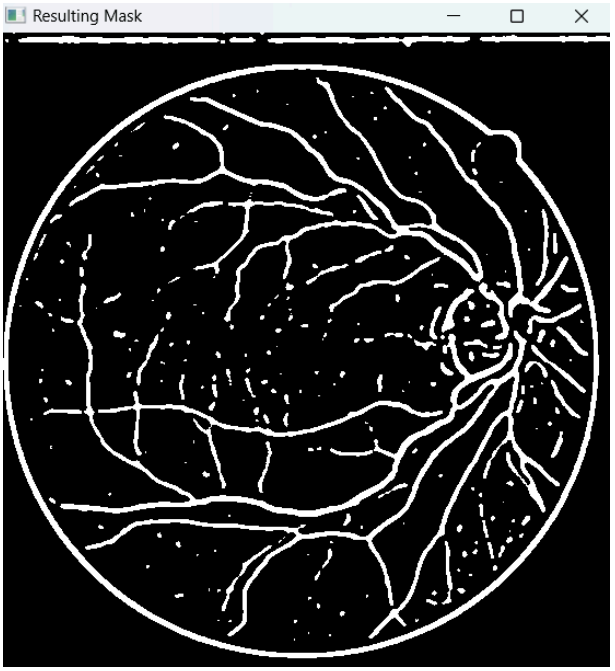
veins_mask: This function takes a grayscale fundus image (g_img) as input and generates a mask highlighting the veins in the image. It uses morphological

operations like dilation, adaptive thresholding, and median blurring to enhance the veins.



Feature Extraction: A sum filter operation is applied to identify prominent features related to the optic disc. Additionally, the brightest pixel within a specified window size is extracted to determine potential disc locations.
Mask of the vein extracted from the colored images:





The Optic Disc Localization:

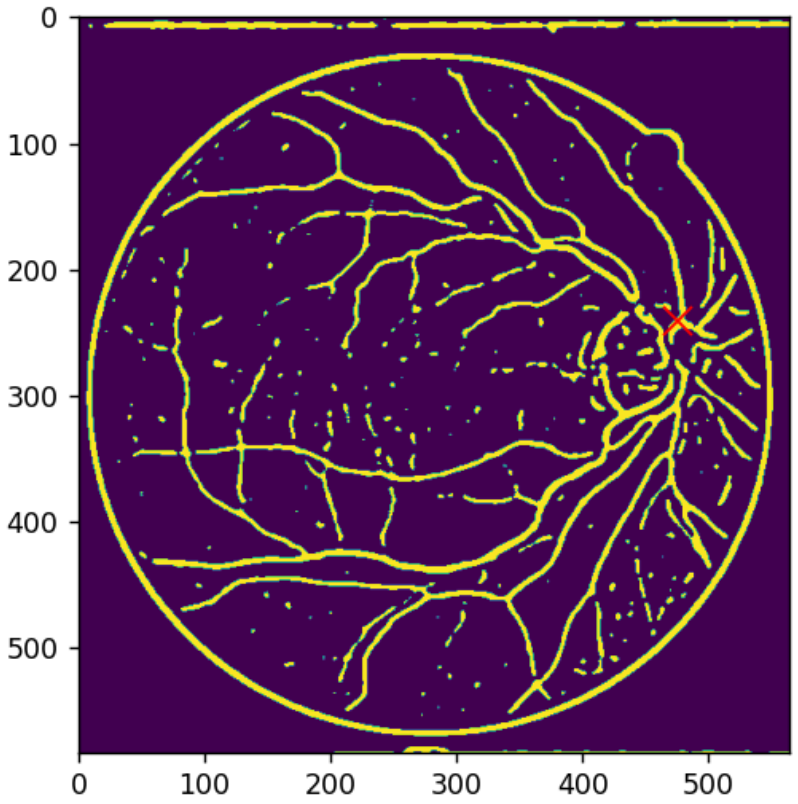
- **Using Veins:**

sum_filter: This function performs a sum filter operation on the input image (img1) using a specified filter size (m). It calculates the sum of pixel values within a moving window of size m x m and replaces each pixel with the sum of its local neighborhood.

max_coordinate: This function finds the coordinates of the maximum value pixel in the filtered image (filt_img).

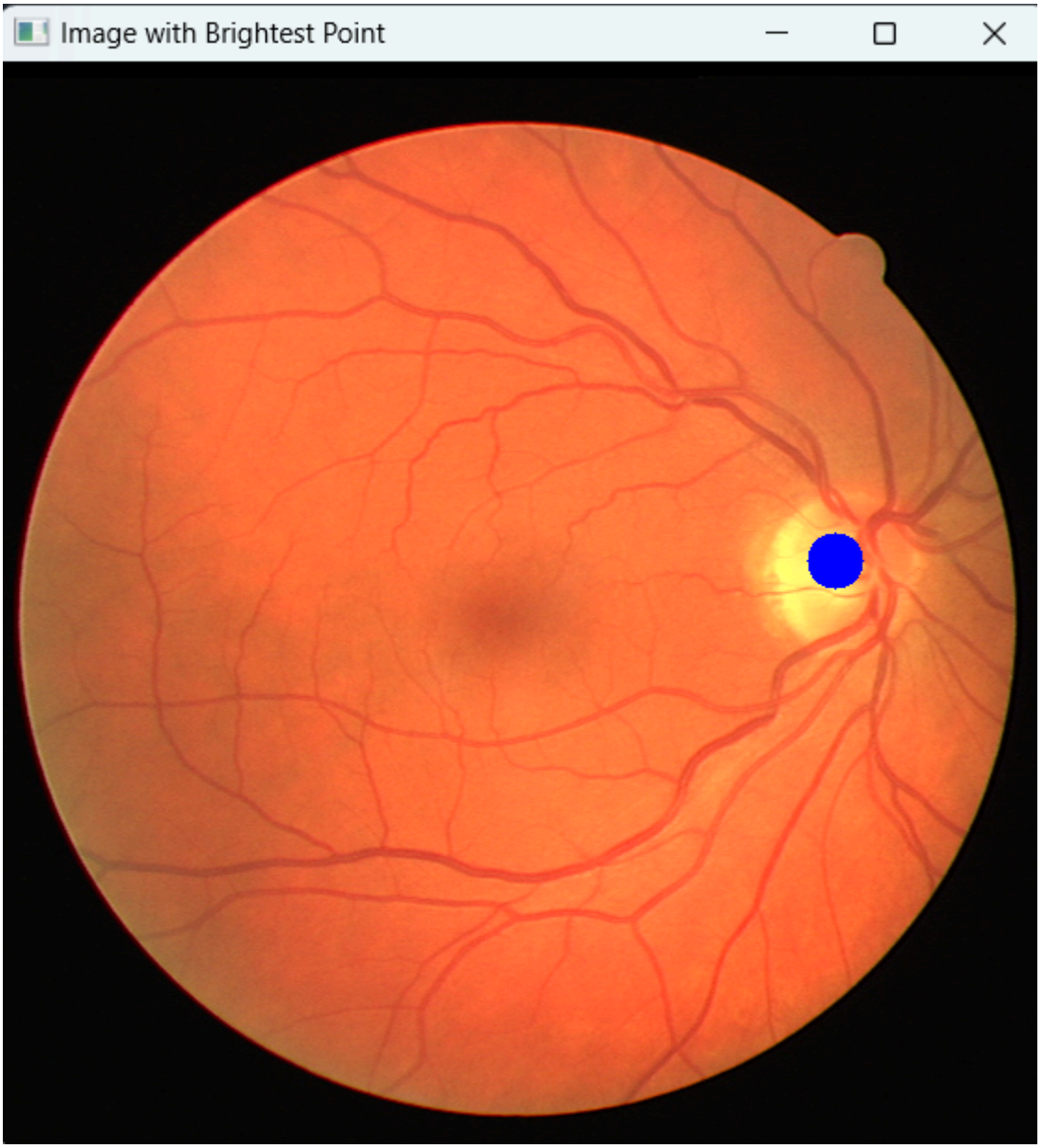
erode_mask: This function performs erosion followed by thresholding on the input image (e_img) to generate a binary mask.

extract_brightest_pixel: This function identifies the brightest pixel in the input image within a specified window size. It iterates through the image, calculates the intensity of each pixel within the window, and returns the coordinates of the brightest pixel.The red mark indicates the centre of the image.



- **Using Brightest Region**

extract_brightest_pixel: This function searches through an image to locate the pixel with the highest intensity within a specified window size. It iterates over each pixel, defines a local region around it, calculates the maximum intensity within that region, and updates the position of the brightest pixel accordingly. Finally, it returns coordinates of the adjusted brightest pixel.



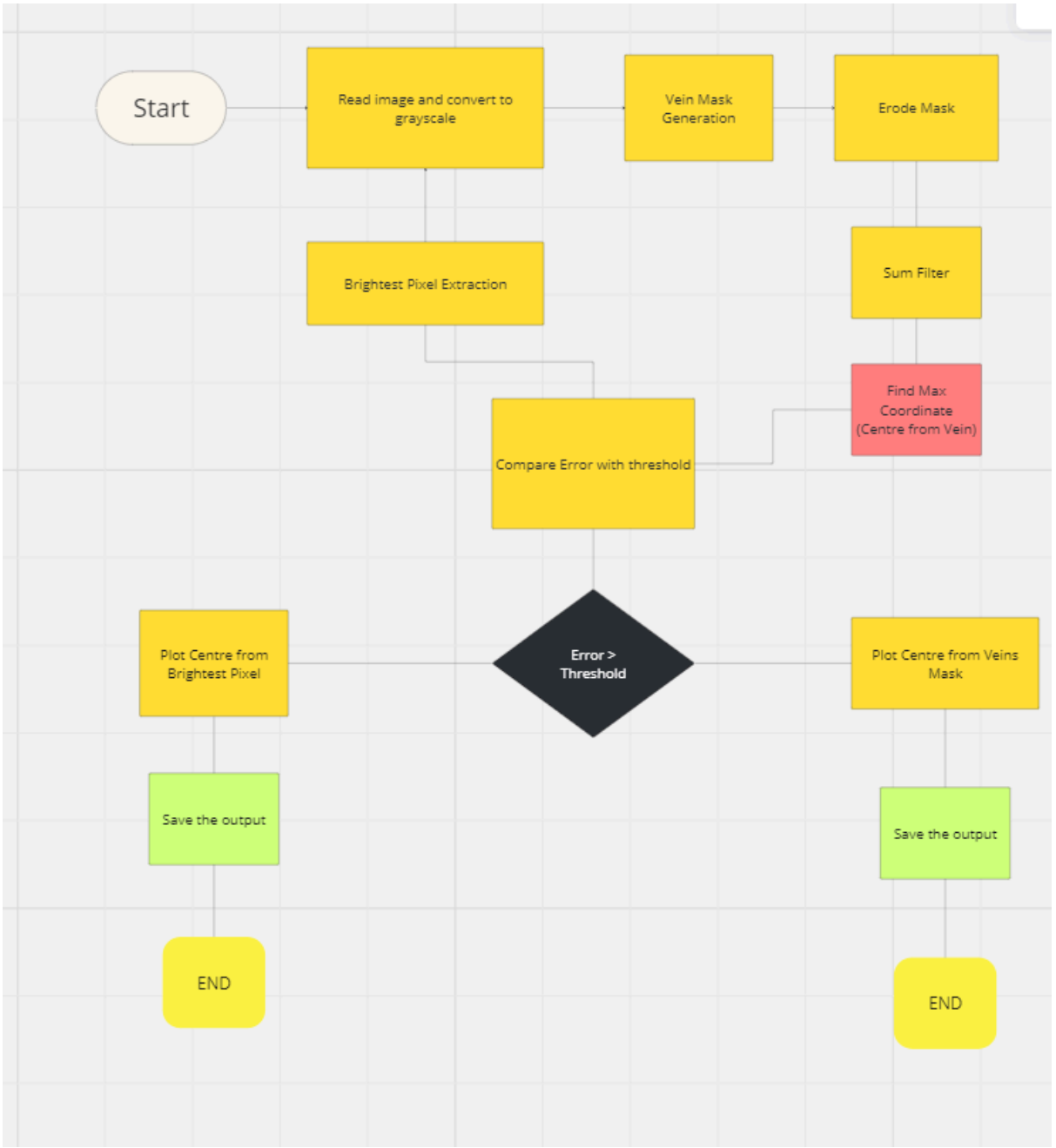
Error Analysis: The distance between the brightest pixel and the calculated vein centre is computed to assess the accuracy of optic disc localization.

- **Error_Calculation:** This function calculates the Euclidean distance between two points.

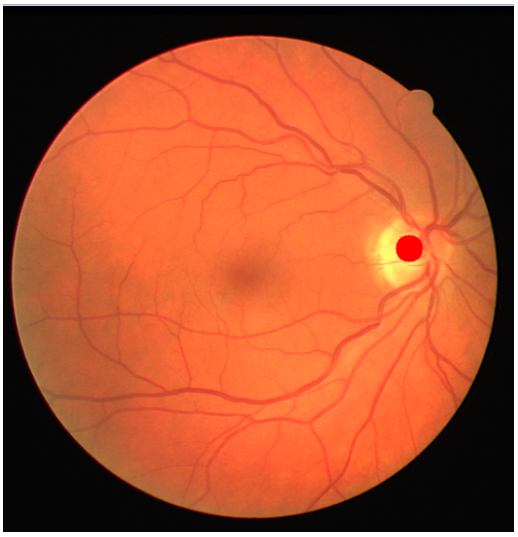
Visualization and Output: The detected features are visualised by overlapping circles on the original images, and the resulting masks are displayed. Output images are saved for further analysis.

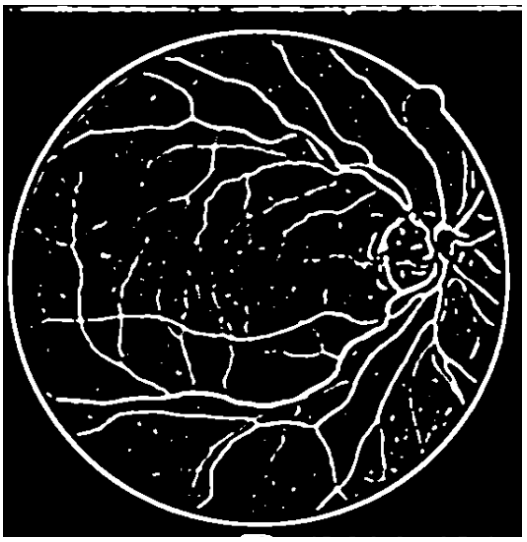
Error Reporting: The errors between the detected and ground truth optic disc centres are recorded and saved in a CSV file for quantitative evaluation.

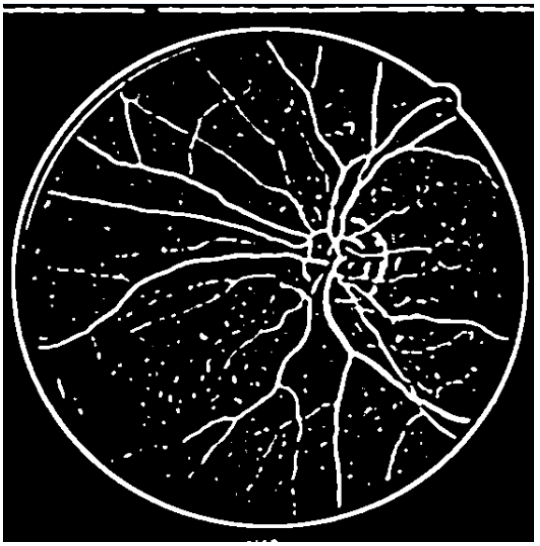
Flow Chart:

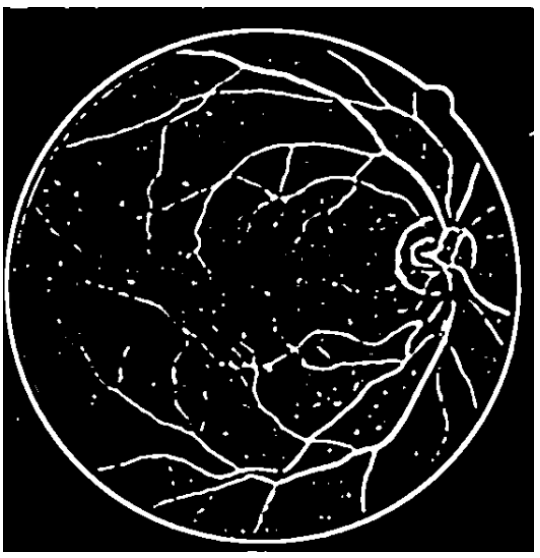


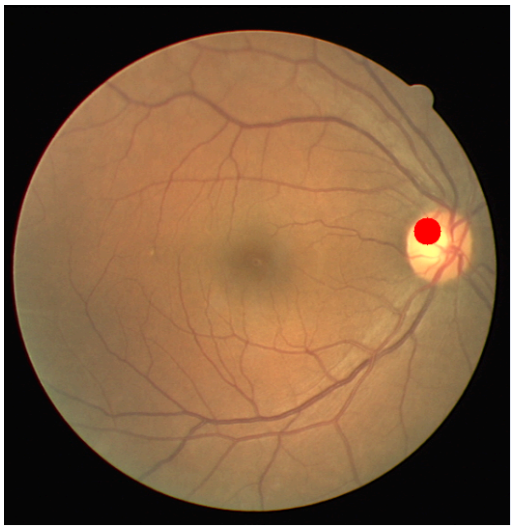
Results:

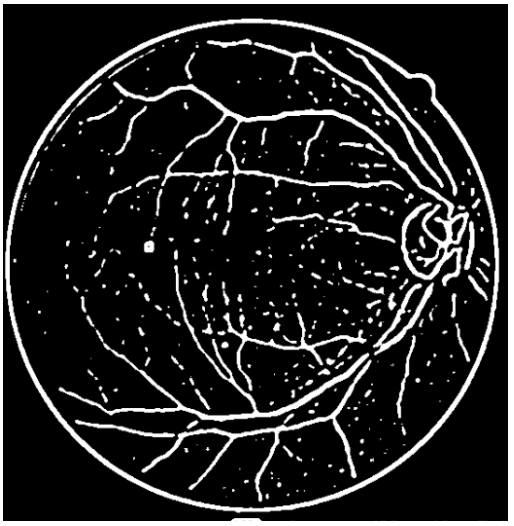


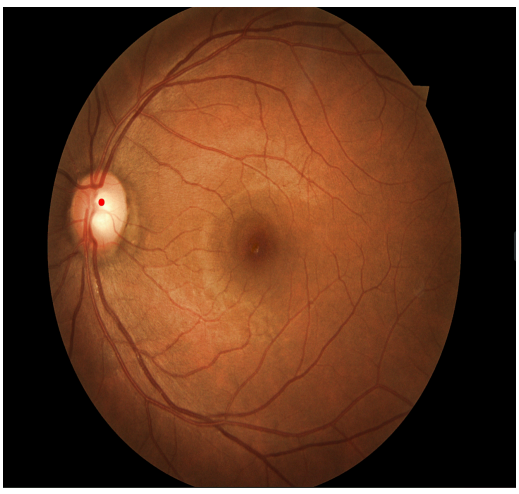


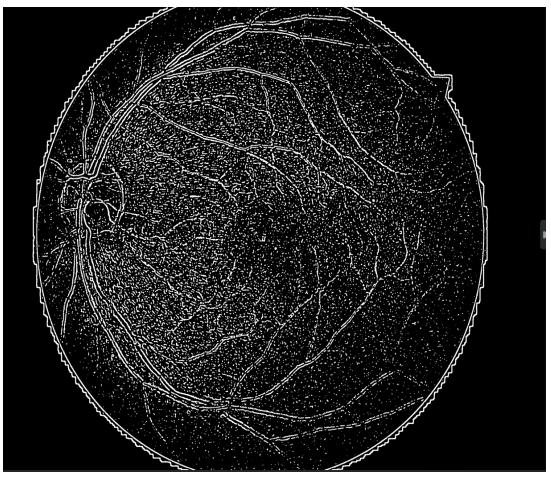












Errors:

Image	Error
02_test.tif	11.40175425
04_test.tif	9.055385138
06_test.tif	20.02498439
08_test.tif	13.60147051
10_left.jpeg	1605.520788
10_right.jpeg	162.0030864
10_test.tif	18.78829423
12_test.tif	21.02379604
13_left.jpeg	57.38466694
13_right.jpeg	64.81512169
13_test.tif	10.63014581
14_test.tif	40.04996879
15_test.tif	13.03840481
16_test.tif	19.23538406
17_left.jpeg	64.19501538
17_test.tif	19.41648784
18_test.tif	14.31782106
19_left.jpeg	60.14149982
19_right.jpeg	81.70679286
19_test.tif	14.86606875

Code:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import math
import os
import pandas as pd

def veins_mask(g_img):
    kernel_dilate = cv.getStructuringElement(cv.MORPH_RECT, (9, 9))
    dilated_img = cv.dilate(g_img, kernel_dilate)
    sub = g_img - dilated_img
    median_blur_img = cv.medianBlur(sub, 5)
    adaptive_thresh = cv.adaptiveThreshold(median_blur_img, 255,
cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 2)
    median_blur_thresh = cv.medianBlur(adaptive_thresh, 5)
    not_img = cv.bitwise_not(median_blur_thresh)
    return not_img

# Finding out the Centre

def sum_filter(img1,m):
    r,c=img1.shape
```

```
    filtered_img=np.zeros_like(img1)
    h = m // 2
    w = m // 2
    padded_img = np.pad(img1, ((h, h), (w, w)), mode='constant')
    for i in range(r):
        for j in range(c):
            roi = padded_img[i:i + m, j:j + m]
            sum_roi = np.sum(roi)#taking the sum of the roi
            filtered_img[i, j] = sum_roi
    return filtered_img

def max_coordinate(filt_img):
    r,c=filt_img.shape
    print(r,c)
    max_pix=filt_img.max()#gives maximum filter
    print(max_pix)
    for i in range(r):
        for j in range(c):
            if filt_img[i,j]==max_pix:
                return i,j

def erode_mask(e_img):
    eroded = cv.erode(e_img, np.ones((5, 5)))
    _, binary_image = cv.threshold(eroded, 127, 1, cv.THRESH_BINARY)
    return binary_image


# Brightest Region
def extract_brightest_pixel(image, window_size):
    max_intensity = 0
    brightest_pixel = None
    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            top_left = (max(0, x - window_size // 2), max(0, y - window_size
// 2))
            bottom_right = (min(image.shape[1], x + window_size // 2),
min(image.shape[0], y + window_size // 2))
            roi = image[top_left[1]:bottom_right[1],
top_left[0]:bottom_right[0]]
            intensity = roi.max()
            if intensity > max_intensity:
                max_intensity = intensity
                brightest_pixel = (x, y)
    # Adjust position to the center of the brightest region
    top_left = (max(0, brightest_pixel[0] - window_size // 2), max(0,
brightest_pixel[1] - window_size // 2))
    bottom_right = (min(image.shape[1], brightest_pixel[0] + window_size //
2), min(image.shape[0], brightest_pixel[1] + window_size // 2))
    brightest_pixel = ((top_left[0] + bottom_right[0]) // 2, (top_left[1] +
bottom_right[1]) // 2)

    return brightest_pixel
```

```
def error_1(pixel_centre,x2,y2):
        x1, y1 = pixel_centre
        error = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
        print("Diif",error)
        if error>100:
            return x1,y1
        else:
            return pixel_centre

def Error_Calculation(x2,y2,x1,y1):
    Error = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    print("The error", Error)
    return Error




input_dir = "Assignment-2-20240428T163732Z-001/Fundus image"
output_dir = "output_images/"
coordinates_file = "Assignment-2-20240428T163732Z-001/optic_disc_centres.csv"
df = pd.read_csv(coordinates_file)
coordinates = {row['image']: (row['x'], row['y']) for _, row in
df.iterrows()}
os.makedirs(output_dir, exist_ok=True)
image_files = os.listdir(input_dir)
errors_output_file = "errors.csv"
errors = []

for filename in image_files:
    if filename in coordinates:
     img_path = os.path.join(input_dir, filename)
     img = cv.imread(img_path)
     g_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

     #Veins Maps
     veins_map = veins_mask(g_img)
     b_img = erode_mask(veins_map)
     img2 = sum_filter(b_img, 18)
     y2, x2 = max_coordinate(img2)

     #Brightest Region/ Pixel
     brightest = extract_brightest_pixel(g_img, 10)

     # Original Centre
     x1, y1 = coordinates[filename]

     #Error
     x_2, y_2 = error_1(brightest, x2, y2)
     error=Error_Calculation(x_2, y_2, x1, y1)
     errors.append({'Image': filename, 'Error': error})

     #Draw Circle and plot
```

```
     cv.circle(img, (brightest[0], brightest[1]), 15, (0, 0, 255), -1)  # Red
dot with radius 15
     plt.plot(x2, y2, marker='x', color='red', markersize=10)
     plt.imshow(veins_map)
     plt.axis('on')

     #save output
     output_path = os.path.join(output_dir, filename)
     cv.imwrite(output_path, img)
     veins_map_output_path = os.path.join(output_dir, "veins_" + filename)
     cv.imwrite(veins_map_output_path, veins_map)
     cv.imshow("Original Image", img)
     cv.imshow("Resulting Mask", veins_map)
     plt.show()
     cv.waitKey(0)

errors_df = pd.DataFrame(errors)
errors_df.to_csv(errors_output_file, index=False)
```