Lab Report 8

Aim

Image segmentation using point segmentation, line segmentation and edge detection.

Theory

Image segmentation is the process of dividing an image into multiple segments or regions, each of which corresponds to a different object or part of the image. Point segmentation, line segmentation, and edge detection are all techniques that can be used to perform image segmentation.

Point segmentation involves identifying individual points or pixels within an image that belong to a particular segment or object. This can be achieved using techniques such as clustering or thresholding, where pixels with similar characteristics are grouped together.

Line segmentation involves identifying linear features within an image, such as edges or contours, and using these features to separate the image into different segments. This can be achieved using techniques such as the Hough transform, which can detect lines or curves within an image.

Edge detection involves identifying abrupt changes in brightness or color within an image, which can be used to locate the boundaries between different objects or segments. This can be achieved using techniques such as the Sobel operator, which highlights areas of the image with high spatial gradients.

All three of these techniques can be used in combination to perform image segmentation. For example, edge detection can be used to identify the boundaries between different segments, while point segmentation and line segmentation can be used to group pixels or linear features within each segment. The choice of technique will depend on the specific requirements of the image segmentation task, as well as the characteristics of the image being segmented.

CODE

```
% POINT SEGMENTATION

fprintf('please Select an image');
y=uigetfile('*.*');
i=imread(y);
k=rgb2gray(i);
d = padarray(k,[1 1],0,'both');
[r,c]=size(d);
s=zeros(r+2,c+2)
```

```
for R =2:(r-1)
   for C=2:(c-1)
      s(R,C)= d(R+1,C)+d(R-1,C)+d(R,C+1)+d(R,C-1)-4*d(R,C);
%         if value>=100
%             d(R,C)=1;
%         else
%             d(R,C)=0;
%         end
   end
end

subplot(1,2,1);imshow(k);title('Original Image');
subplot(1,2,2);imshow(s);title('detected');
```

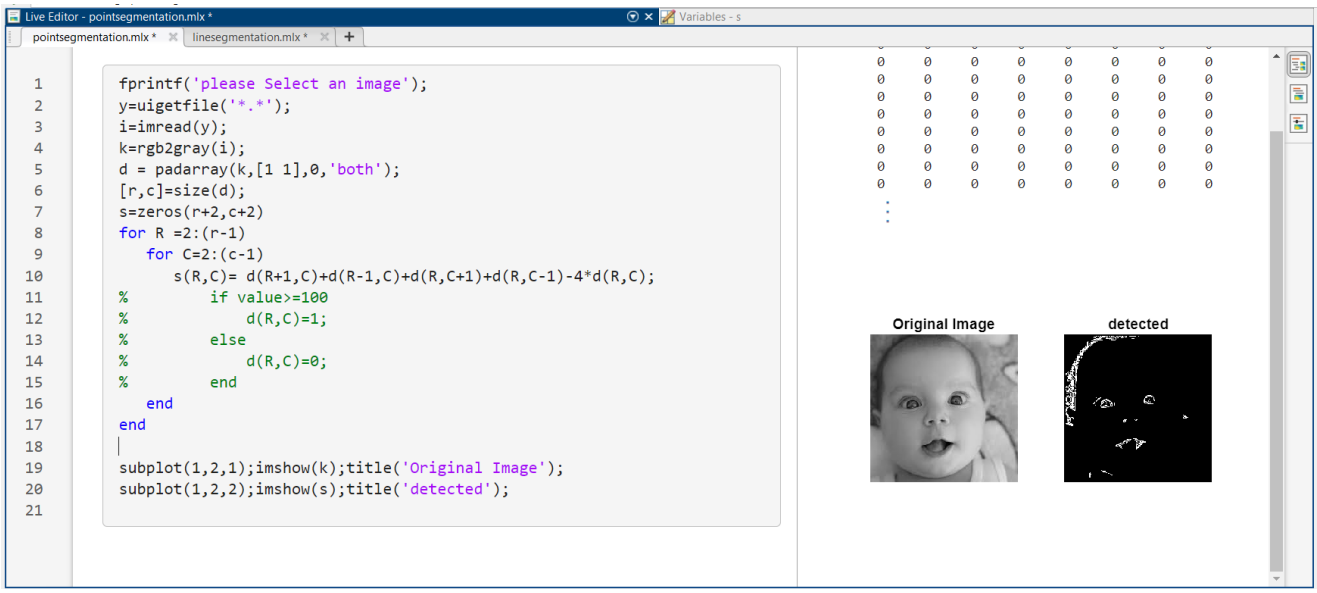

Figure 1: use of point segmentation to detect points in image in MATLAB

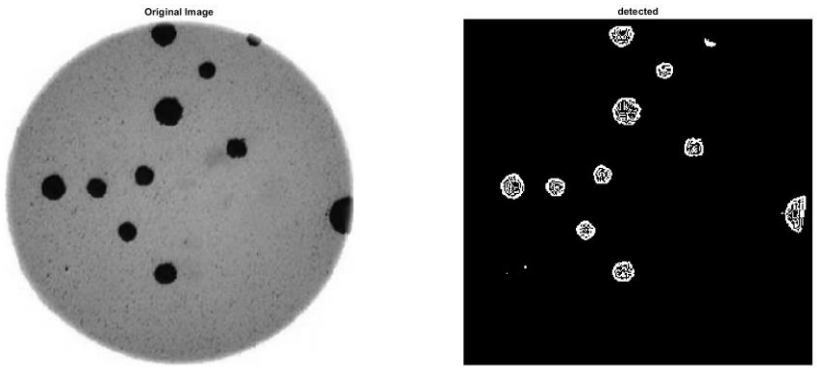

Figure 2: point detection using point segmentation in MATLAB

CODE

```
% LINE SEGMENTATION
fprintf('please Select an image');
y=uigetfile('*.*');
i=imread(y);
k=double(rgb2gray(i));
d = padarray(k,[1 1],0,'both');
[r,c]=size(d);
hori=zeros(r+2,c+2)
verti=zeros(r+2,c+2)
diag1=zeros(r+2,c+2)
diag2=zeros(r+2,c+2)
hor=[-1 -1 -1 ; 2 2 2 ; -1 -1 -1 ] % horizontal filter
ver=[-1 2 -1 ; -1 2 -1 ; -1 2 -1 ] % Vertical filter
dia1=[2 -1 -1 ; -1 2 -1 ; -1 -1 2 ] % diagonal 45 degree filter
dia2=[-1 -1 2 ; -1 2 -1 ; 2 -1 -1 ] % diagonal 135 degree filter
for x =2:(r-1)
   for y=2:(c-1)
      kernel= [d(x-1,y-1) d(x-1,y) d(x-1,y+1) ; d(x,y-1) d(x,y)
d(x,y+1) ; d(x+1,y-1) d(x+1,y) d(x+1,y+1)];
      h=sum(sum(hor.*kernel));
      v=sum(sum(ver.*kernel));
      d1=sum(sum(dia1.*kernel));
      d2=sum(sum(dia2.*kernel));
      hori(x,y)=h;
      verti(x,y)=v;
      diag1(x,y)=d1;
      diag2(x,y)=d2;
   end
end
all=hori+verti+diag1+diag2;
subplot(2,3,1);imshow(i);title('Original Image');
subplot(2,3,2);imshow(hori);title('horizontal lines');
subplot(2,3,3);imshow(verti);title('vertical lines');
subplot(2,3,4);imshow(diag1);title('diagonal line 45');
subplot(2,3,5);imshow(diag2);title('diagonal line 135');
subplot(2,3,6);imshow(all);title('all line in one');
```

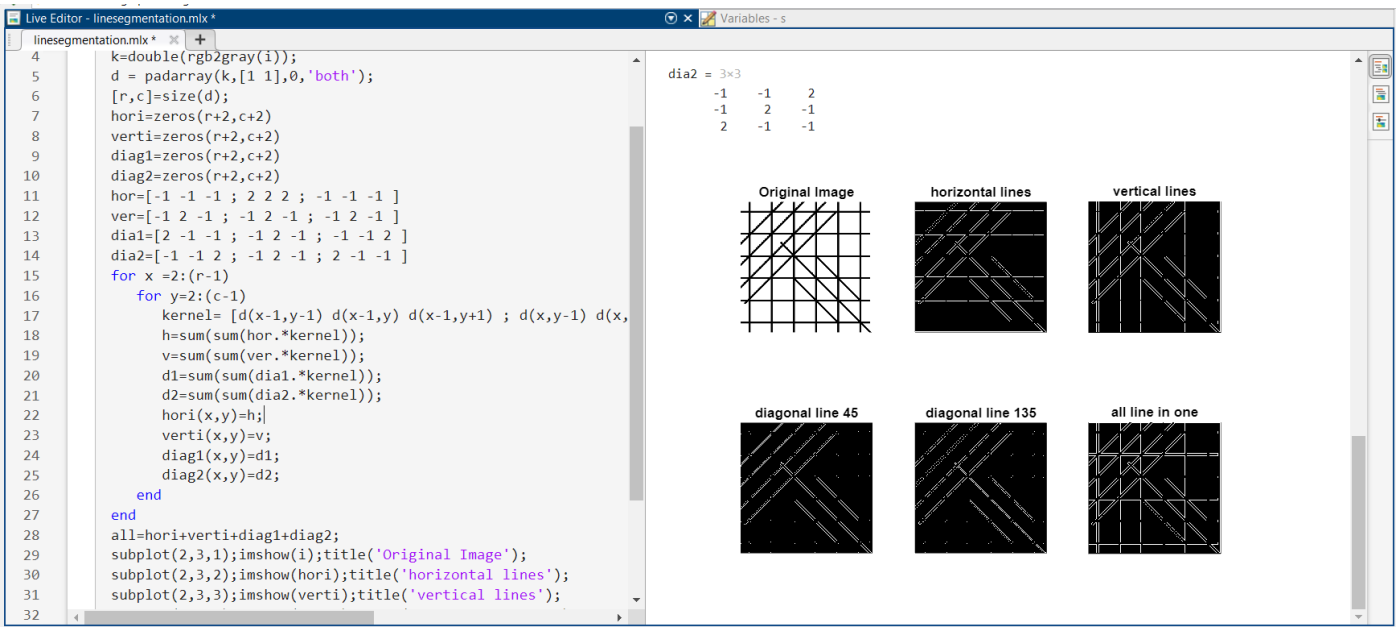

Figure 3: use of line segmentation to detect lines in MATLAB

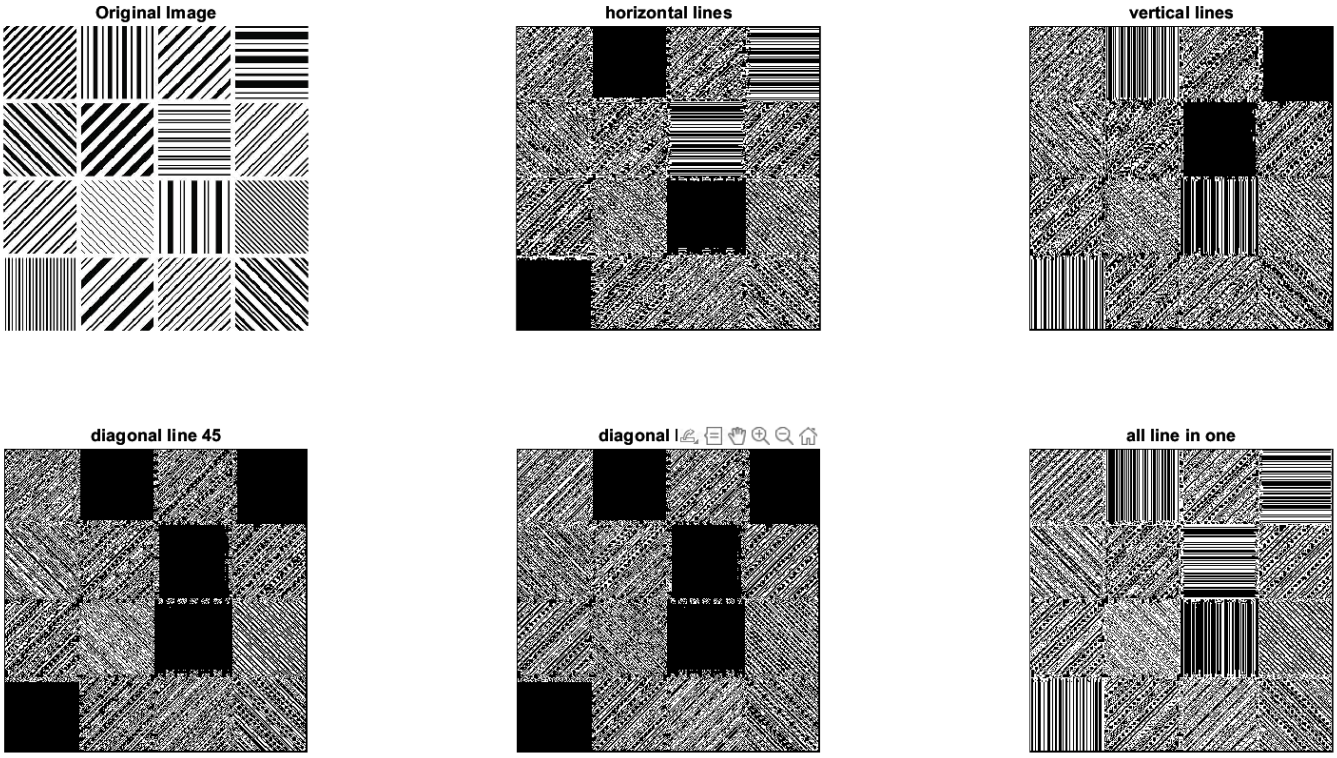

Figure 4: detection of various types of lines in MATLAB

EDGE DETECTION

Edge detection is a computer vision technique that involves identifying the boundaries of objects within an image. It is a crucial step in many image processing applications, such as object recognition, image segmentation, and feature extraction. Edge detection algorithms typically work by analyzing the variations in

brightness, colour, or texture across an image. The most commonly used edge detection algorithms include the Sobel, Prewitt, and Canny edge detection algorithms. The Sobel and Prewitt algorithms use convolution filters to identify edges based on the intensity gradients in the image. The Canny algorithm, on the other hand, uses a more sophisticated approach that involves detecting edges at multiple scales and suppressing false positives. Once the edges have been detected, they can be further processed to extract useful information, such as the shape and size of objects in the image. This information can then be used for a wide range of applications, such as object tracking, face recognition, and autonomous driving.

CODE

```
% EDGE DETECTION
fprintf('please Select an image');
y=uigetfile('*.*');
i=imread(y);
J=rgb2gray(i);



K=padarray(J,[1,1],0);
K=double(K);
[rows,columns]=size(K);
L=zeros(rows,columns);
gx=[-1 -1 -1;0 0 0;1 1 1];
gy=[-1 0 1;-1 0 1;-1 0 1];
s1=[0 0 0;0 0 0;0 0 0];
s3=[0 0 0;0 0 0;0 0 0];
for r=2:rows-1
    for c=2:columns-1
      kernel=[K(r-1,c-1) K(r-1,c) K(r-1,c+1); K(r,c-1) K(r,c)
K(r,c+1); K(r+1,c-1) K(r+1,c) K(r+1,c+1)];
      s1=kernel.*gx;
      s2=sum(s1,"all");
      s3=kernel.*gy;
      s4=sum(s3,"all");
      L(r,c)=sqrt(s2.^2+s4.^2);
    end
end
subplot(1,2,1);imshow(i);title("Original image");
subplot(1,2,2);imshow(uint8(L),[]);title('edges detected');
```

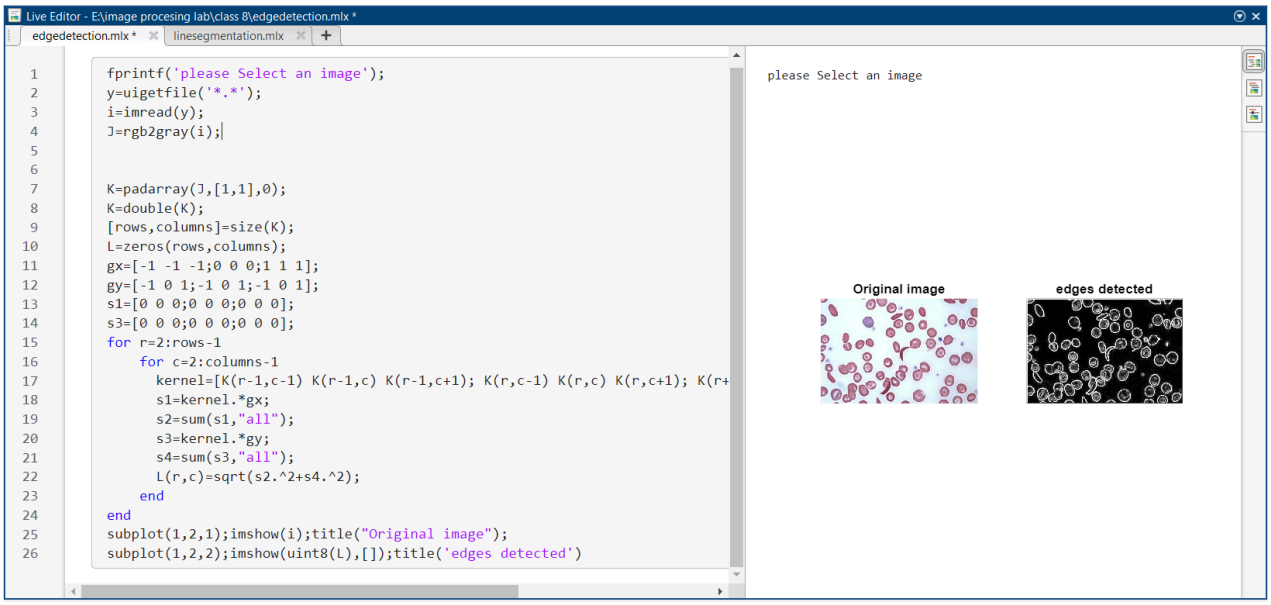

Figure 5: use of edge detection code to detect edges in MATLAB

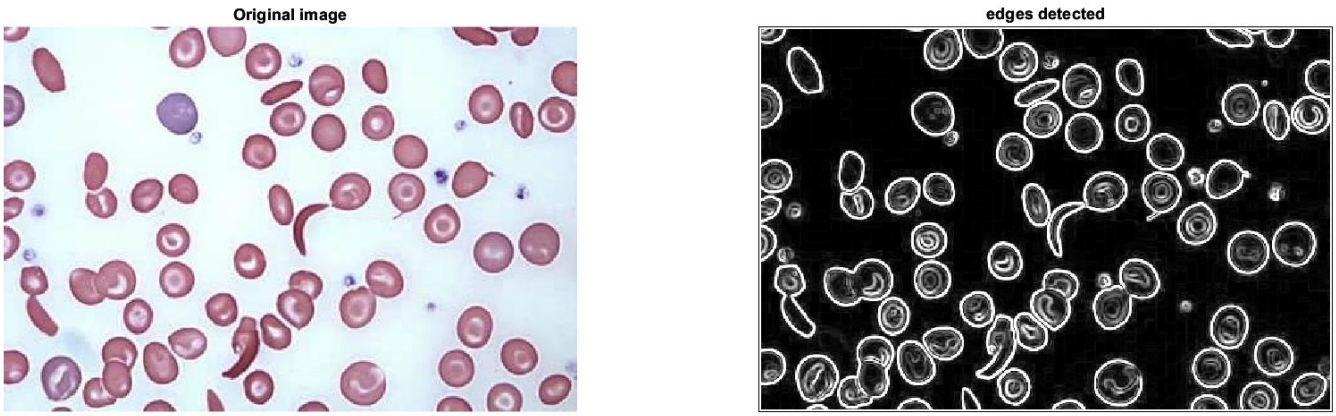

Figure 6 edge detection *in MATLAB*

THRESHOLD BASED SEGMENTATION

Threshold-based segmentation is a commonly used technique in image processing and computer vision for segmenting an image into different regions based on their intensity values. The basic idea behind threshold-based segmentation is to set a threshold value, and then classify each pixel in the image based on whether its intensity value is above or below the threshold.

The threshold value can be chosen manually or automatically, depending on the specific application and the characteristics of the image being segmented. If the threshold is chosen manually, it is typically based on some prior knowledge of the image or the desired segmentation result. If the threshold is chosen automatically, there are several methods that can be used, such as Otsu's method, which selects a threshold that minimizes the variance between the two classes of pixels (above and below the threshold).

Once the threshold value is set, each pixel in the image is compared to the threshold, and is classified as either part of the foreground (above the threshold) or part of the background (below the threshold). The result is a

binary image, where the foreground pixels are represented as white and the background pixels are represented as black.

Threshold-based segmentation is a simple and computationally efficient technique, but it has some limitations. It works well when there is a clear contrast between the foreground and background, but may fail when there is significant overlap in intensity values between the two classes. In such cases, more advanced techniques such as edge detection and region-growing may be necessary.

CODE

```
% THRESHOLD-BASED SEGMENTATION
fprintf('please Select an image');
y=uigetfile('*.*');
i=imread(y);
d=rgb2gray(i);
[r,c]=size(d);
%Here the histogram is displayed
funchist=imhist(d);
subplot(1,1,1);bar(funchist);title('Histogram of the image');grid on;
%Here the threshold values are set.
s=zeros(r,c);
for R =1:r
   for C=1:c
      if d(R,C)==91 || d(R,C)==84 ||d(R,C)==122
          s(R,C)=1;
      else
          s(R,C)=0;
      end

   end
end

subplot(1,3,1);imshow(d);title('Original Image');
subplot(1,3,2);imshow(s);title('detected');
subplot(1,3,3);imshow(double(d).*s);title('segmented');
```

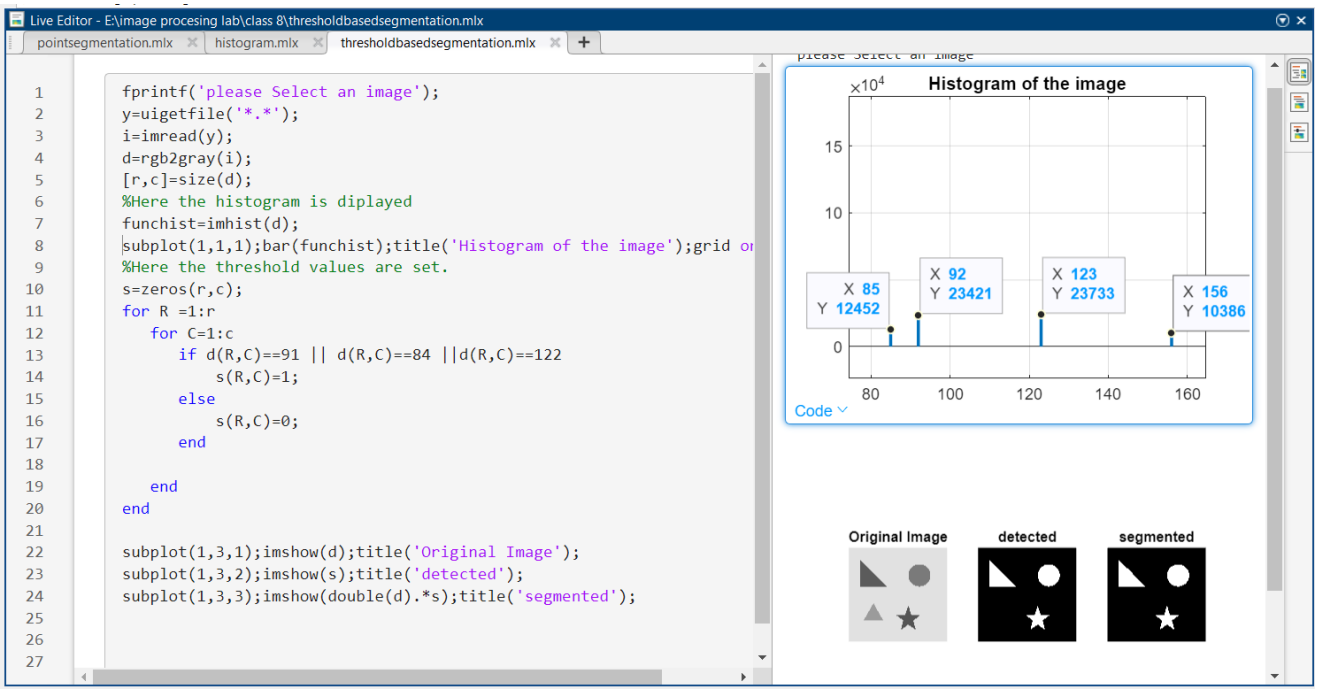

Figure 7: use of threshold-based segmentation in MATLAB

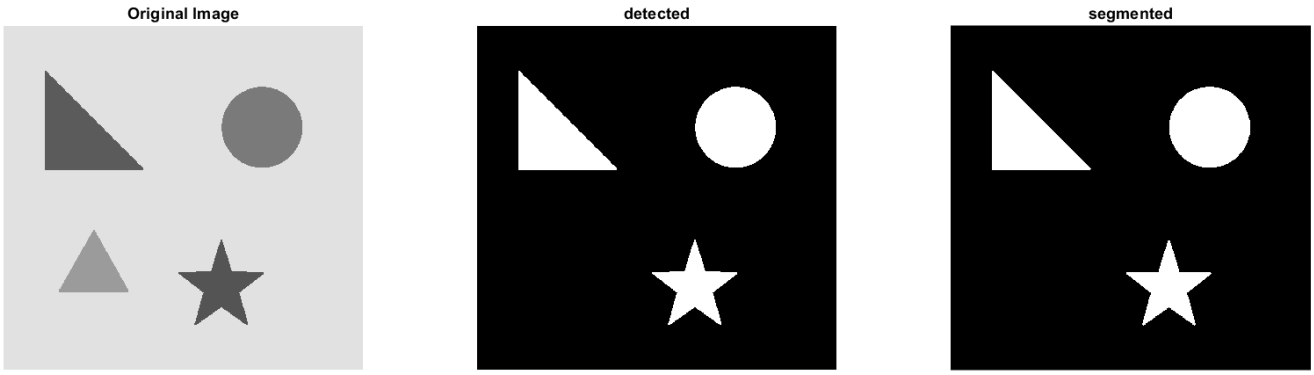

Figure 8: segmented image in MATLAB

REGION BASED SEGMENTATION

Region-based segmentation is a type of image segmentation technique that involves partitioning an image into multiple regions or segments based on certain characteristics or features of the image. The goal of region-based segmentation is to identify and separate different objects or regions within an image based on their similarities or differences.

There are several approaches to region-based segmentation, including thresholding, edge detection, and clustering. In thresholding, the image is converted into a binary image by selecting a threshold value that separates the image pixels into foreground and background. In edge detection, the boundaries of different regions are detected based on changes in intensity or colour. Clustering algorithms group similar pixels together based on their features or attributes, such as colour, texture, or intensity.

Once the image has been segmented into regions, further analysis can be performed on each region separately. This can include feature extraction, object recognition, and classification. Region-based segmentation is

widely used in applications such as image processing, computer vision, and medical imaging, where identifying and analysing specific regions of an image is important.

REGION GROWING SEGMENTATION

Region growing is a type of region-based segmentation technique that involves starting with a seed point or region and gradually expanding the region by including neighbouring pixels or regions that have similar properties or characteristics. The process continues until the entire region of interest has been segmented. The region growing algorithm works by selecting a seed point or region and then examining its neighbouring pixels or regions. The algorithm checks if the neighbouring pixels or regions meet a certain similarity criterion, such as having similar colour, intensity, or texture. If they do, they are added to the growing region, and the algorithm continues to examine their neighbouring pixels or regions. This process continues until no more pixels or regions meet the similarity criterion, and the growing region is complete.

One advantage of region growing is that it can handle images with variable illumination and shading. However, the algorithm can be sensitive to the choice of seed point or region, as the resulting segmented region can vary depending on the starting point. To address this issue, multiple seed points can be used, and the final segmented region can be obtained by merging the results of multiple regions growing processes. Region growing is widely used in medical imaging applications, such as segmenting tumours in MRI or CT scans. It is also used in computer vision applications, such as object recognition and tracking.

CODE

```
 FUNCTION FOR REGION GROWING SEGMENTATION


function [segmented_image] = region_growing(image, seed_point,
threshold)
% Inputs:
%   - image: The input grayscale image
%   - seed_point: A 2-element vector containing the (x,y) coordinates
of the seed point
%   - threshold: The threshold value for region growing
% Output:
%   - segmented_image: The output binary segmented image


% Initialize the segmented image to all zeros
segmented_image = zeros(size(image));


% Get the size of the image
[rows,cols] = size(image);


% Initialize the queue with the seed point
queue = [seed_point(1), seed_point(2)];
```

```
% Loop through the queue until it is empty
while ~isempty(queue)
    % Pop the first element from the queue
    current_point = queue(1,:);
    queue(1,:) = [];

    % Check if the current point is already segmented
    if segmented_image(current_point(2), current_point(1)) == 1
        continue;
    end

    % Check if the current point is within the image boundaries
    if current_point(1) < 1 || current_point(1) > cols || ...
            current_point(2) < 1 || current_point(2) > rows
        continue;
    end

    % Check if the intensity of the current point is below the
threshold
    if image(current_point(2), current_point(1)) < threshold
        continue;
    end

    % Mark the current point as segmented
    segmented_image(current_point(2), current_point(1)) = 1;

    % Add the neighbors of the current point to the queue
    queue(end+1,:) = [current_point(1)-1, current_point(2)-1];
    queue(end+1,:) = [current_point(1), current_point(2)-1];
    queue(end+1,:) = [current_point(1)+1, current_point(2)-1];
    queue(end+1,:) = [current_point(1)-1, current_point(2)];
    queue(end+1,:) = [current_point(1)+1, current_point(2)];
    queue(end+1,:) = [current_point(1)-1, current_point(2)+1];
    queue(end+1,:) = [current_point(1), current_point(2)+1];
    queue(end+1,:) = [current_point(1)+1, current_point(2)+1];
end
```

CODE

```
 REGION GROWING SEGMENTATION

fprintf('please Select an image');
y=uigetfile('*.*');
i=imread(y);
image=rgb2gray(i);
subplot(1,2,1);
imshow(image);title("Original Image");
% Set the seed point and threshold value
seed_point = [300    , 550];
threshold = 10;
% Call the region growing function
segmented_image = region_growing(image, seed_point, threshold);
% Display the segmented image
subplot(1,2,2);
imshow(segmented_image);
```

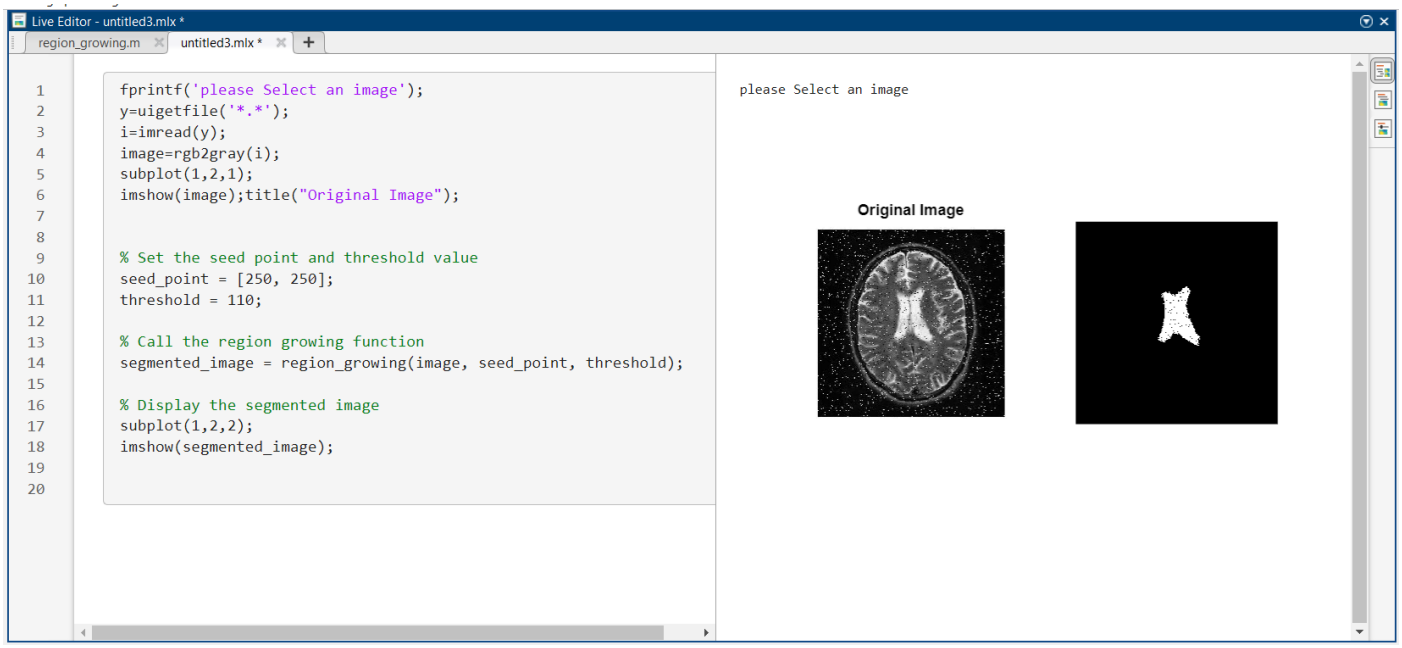

Figure 9: use of region growing segmentation in MATLAB

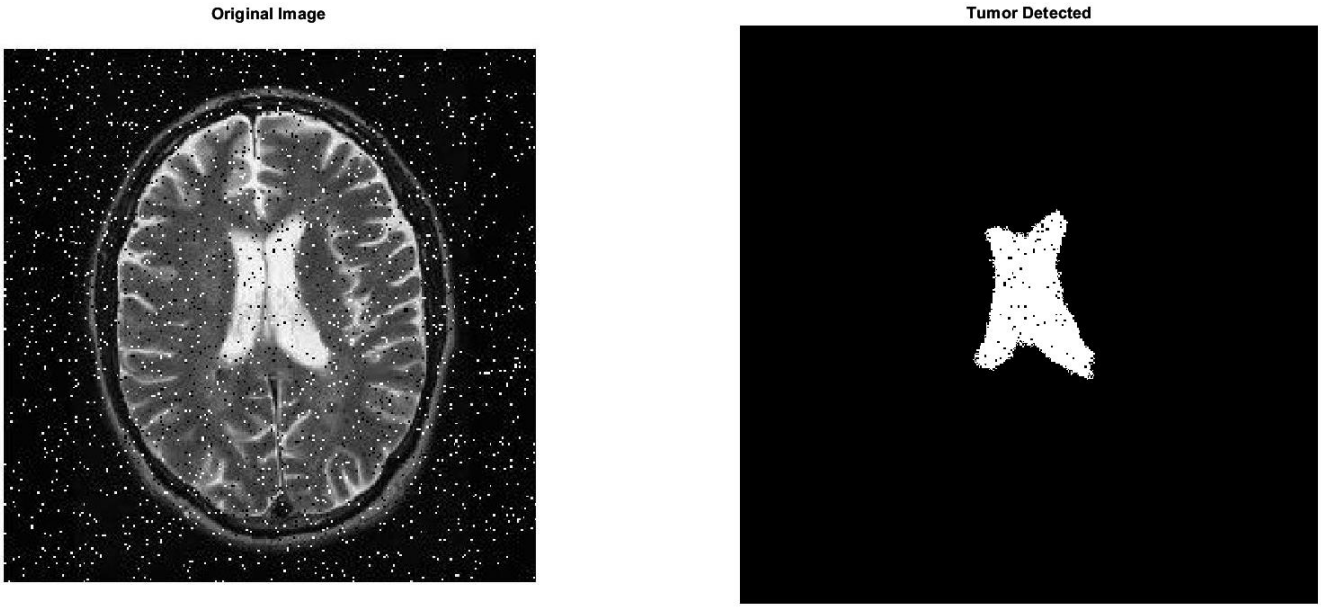

Figure 10: segmented image in MATLAB

Shreenandan Sahu |120BM0806