# LAMPIRAN

**Region Growing.m**

```
function [segmented_image, region_total, e]=region_growing(I, threshold, neighbours )
%%
% This function will segment an image using region growing method with one
% seed point in (1,1) and mean as the threshold.
% INPUT
%   I = Grayscale image to be segmented
%   threshold = Positive integer representing the threshold of regions
%   neighbours = Positive integer representing the neighbours computation (
%   4 or 8 )
% OUTPUT
%   segmented_image = Segmented image
%   region_total = Total region generated
%   e = Computation time
%
% EXAMPLE OF USAGE
%   Suppose one wants to segment an image, I, using region growing method
%   with 50 threshold, and 4-connected neighbourhood.
%   The callback of this function can be as follows:
%       region_growing( I, 50 , 4 )
%
% This function will utilize dynamic array to store the queue. Another
% function utilizing static array is also implemented. The reason is to see
% the comparisan of computation time between them.
%%
% Start the cpu clock
t = cputime;
% Extract image size
[i_height, i_width ] = size(I);
i_size = i_height * i_width;

% Initiate a temporary matrix for output image, and the regions details.
I_temp=zeros(i_height, i_width);
region_total = 1;
region_size = 1;

% Set the seed point
seed_point = [1,1];
% Initatiate mean to be the intensity of the seed point
u_ri= double(I(seed_point(1), seed_point(2) ));

%Initiate a queue to store the execution sequence
queue = [];
queue_last = 1;
queue(1,:) = seed_point;

% Set the neighbours matrix
if( neighbours == 8 )
   neighbours = [-1 0; -1 1;0 1; 1 1;1 0; 1 -1;-1 0;-1 -1 ];
else
   neighbours = [-1 0; 0 1; 1 0; -1 0 ];
end

% Start growing the region sequentially
for im_counter_w = 1 : i_width
```

```
for im_counter_h = 1 : i_height%Calculate the computation time
   % Check whether the pixel has ever been evaluated before
   if( I_temp(im_counter_h, im_counter_w) == 0 )
      % Check whether the pixel is within the same region
      same_region = abs ( double(I(im_counter_h, im_counter_w)) - double(u_ri)) <= threshold;

      % Generate a new queue if the region is not the same
      if ~same_region
         %Clear up the queue, and initiate all the parameter again
         %for the new region
         queue = [];
         queue(1,:) = [im_counter_h, im_counter_w];
         region_size = 1;
         queue_last = 1;
         u_ri = I(im_counter_h, im_counter_w);
         % Increase number of region
         region_total = region_total + 1;
      else
         % If it is the same region, labelled the pixel
         I_temp(im_counter_h, im_counter_w ) = region_total;
      end
      % Start exploring the neighbour until it reaches the end of the
      % queue or the region size is the same as the image size
      while queue_last <= region_size && region_size <= i_size
         % Get the index of the next pixel from the queue
         i = queue(queue_last, 1);
         j = queue(queue_last, 2);
         % Labelled this pixel to be on the same region
         I_temp(i,j ) = region_total;

         % Start exploring the neighbours
         for n = 1 : size(neighbours)
            neighbour_position = [i j] + neighbours(n,:);

            %Check if it is stil within image
            if neighbour_position(1)>= 1 && neighbour_position(2)>=1....
                  && neighbour_position(1) <= i_height && neighbour_position(2)<= i_width

               % Check whether it falls within the same region
               neighbour_intensity = I(neighbour_position(1), neighbour_position(2));
               same_region = abs ( double(neighbour_intensity) - double(u_ri) ) <= threshold;
               %If it is in the same region and it is not labelled
               %yet
               if same_region
                  if(  I_temp(neighbour_position(1),neighbour_position(2)) == 0)
                     % Update the queue list
                     queue = [ queue; [neighbour_position(1),neighbour_position(2)] ];
                     % Labelled the pixel
                     I_temp( neighbour_position(1), neighbour_position(2) ) = region_total;
                     % Update the mean value and increase the
                     % region size
                     u_ri = ( double(u_ri*region_size) + double(neighbour_intensity) ) / (region_size+1);
                     region_size = region_size + 1;
                  end
               end
            end
```

```
            end
            %Go to the next qeue
            queue_last =  queue_last +1 ;
          end
        end

    end
end

%Labelled the image
segmented_image = label2rgb(uint8(I_temp));
%Calculate the computation time
e = cputime - t;
end
```