**PRAKTIKUM 10**

**Image Segmentation**

**SISTEM PENGOLAHAN CITRA**

**PROGRAM STUDI SISTEM KOMPUTER**

**SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY**

**UNIVERSITAS PELITA HARAPAN**

**DISUSUN OLEH:**

**Alfa Satya Putra, B.Sc., M.Sc.**

**Intro**

This laboratory illustrates the concepts of pixel neighborhoods and connected pixel sets. We will denote the set of 2-D lattice points by S and individual lattice points by s ∈ S. When necessary will we explicitly denote the 2-D coordinates of a lattice point by s = (s1,s2), where s1 is the horizontal coordinate (column) and s2 is the vertical coordinate (row). The upper left-most pixel of the image will correspond to lattice location s = (0, 0).

In this laboratory, we will use following definitions for neighborhood and connectedness:

* We will use **4 point neighborhood**, neighbors of lattice point (s1,s2) are:

**∂(s1,s2) = {(s1 − 1,s2),(s1 + 1,s2),(s1,s2 − 1),(s1,s2 + 1)}**

* We will use **free boundary**, pixels along boundary of image have less than 4 neighbors each
* Two neighboring lattice points r ∈ ∂s said to be connected neighbors if **|xs − xr| ≤ T**, xs is pixel value at lattice point s and T is fixed threshold. We will denote connected neighbors of s by the set c(s) ⊂ ∂s. More specifically, for this application: **c(s) = {r ∈ ∂s| |xs − xr| ≤ T}**
* Pixels s and r are said to be connected if there is a sequence of M pixels s1,s2, · · · ,sM such as **s ∈ c(s1), s1 ∈ c(s2), · · ·, sM−1 ∈ c(sM), sM ∈ c(r)**

**Part 1 – Area Fill**

In this section, you will write a program that fills in an area of connected pixels in an image. To do this, you will compute the set of all pixels which are connected to a specified pixel s.

Tasks:

1. Write function **ConnectedNeighbors** to find connected neighbors of a pixel s. The structure of subroutine call should be as follows:

struct pixel { int m,n; /\* m=row, n=col \*/ }

void ConnectedNeighbors(

struct pixel s,

double T,

unsigned char \*\*img,

int width,

int height,

int \*M,

struct pixel c[4])

Subroutine inputs:

* struct pixel s - Contains location of pixel s whose connected neighbors will be computed
* double T - Threshold used in equation (in the intro)
* unsigned char \*\*img - 2-D array of pixels img[m][n], denoted xs in equation
* int width - Width of img[height][width]
* int height - Height of img[height][width]

Subroutine outputs:

* int \*M - pointer to number of neighbors connected to pixel s
* struct pixel c[4] - array containing M connected neighbors to pixel s, M always <= 4

1. Write function called **ConnectedSet** to find all the pixels connected to s0, which may be implemented by maintaining 1) a list of pixels B which are known to be connected to s0, but have not yet been searched, and 2) a segmentation image Ys which is equal to 1 for pixel’s which are known to be connected to s and 0 otherwise. Pseudocode:

Initialize Yr = 0 for all r ∈ S

ClassLabel = 1

ConnectedSet(s0,Y,ClassLabel) {

B ← {s0}

While B is not empty {

s ← any element of B

B ← B − {s}

Ys ← ClassLabel

B ← B S {c(s) ∩ {r : Yr = 0}} }

return(Y)

}

And structure:

void ConnectedSet(

struct pixel s,

double T,

unsigned char \*\*img,

int width,

int height,

int ClassLabel,

unsigned int \*\*seg,

int \*NumConPixels)

Subroutine inputs:

* struct pixel s - contains location of pixel s that will serve as the seed
* double T - This is the threshold used in equation
* unsigned char \*\*img - 2-D array of pixels img[m][n] denoted as xs in equation
* int width - Width of img[height][width].
* int height - Height of img[height][width].
* int ClassLabel - Integer value to label any pixel connected to s.

Subroutine outputs:

* unsigned int \*\*seg - 2-D array of integers, contains class of each pixel and passed to ConnectedSet from the main routine. If a pixel at location i, j is found to be connected to s, then seg[i][j]← ClassLabel. Otherwise the value of seg[i][j] is left unchanged
* int \*NumConPixels –Number of pixels found to be connected to s

1. Download image **img22gd2.tif**. Apply ConnectedSet to extract connected set of pixels for **s = (67, 45)** with varying T, namely **T=1, T=2,T=3**. 67 is the column index and 45 is the row index. For each threshold value, generate an image with the elements in the connected set printed as black and remaining pixels printed as white. Put your **full code** and **images** in the lab report.

**[Bonus] Part 2 – Image Segmentation**

In this section, you will use function for region filling to segment image into connected components.

Tasks:

1. Use function **ConnectedSet** from part 1 to extract all connected sets in the image img22gd2.tif. You can do this by indexing through the image in raster order and applying ConnectedSet subroutine at each pixel that does not yet belong to a connected set. Note that for a small threshold, the size of most connected sets for this image will be small, resulting in large number of connected sets in the segmentation.
2. Generate a segmentation of the image consisting of connected sets containing greater than 100 pixels. Number each of these large connected sets sequentially starting at 1. All remaining connected sets should be labeled as 0. There will be fewer than 255 large connected sets, so you can store the label for each pixel as a 2-D unsigned character array. Save this 2-D array as a monochrome TIFF image, **segmentation.tif**.
3. To view your segmentation clearly, you will need to scramble the colormap to provide contrast between the distinct regions. You can do this in Matlab with the following commands:

x=imread(’segmentation.tif’);

N=max(x(:));

image(x) colormap(rand(N,3))

axis(’image’)

Print or export this color segmentation of the image and include it in your lab report.

References:

* <https://www.gnu.org/software/octave/>
* GNU Octave Manual
* Class Materials, Slide Week 14 & 15
* Purdue ECE 637 Connected Component Lab: https://engineering.purdue.edu/~bouman/grad-labs/Connected-Component-Analysis/pdf/lab.pdf