НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

**Лабораторна робота №7**

з дисципліни «Аналіз зображень та візуалізація в діагностиці та лікуванні»

на тему: «Методи сигментації зображень, засновані на формуванні однорідних ділянок»

Варіант 3

Виконав:

студент гр. БС-41м

Григоревський Є.С.

Перевірила:

Алхімова С.М.

Київ – 2014

**Завдання**

1. Ознайомитися з теоретичними відомостями алгоритмів сигментації зображень, засновані на формуванні однорідних ділянок
2. Відповідно до свого варіанту завантажити та відобразити медичне зображення та створити подію при обробці якої проводиться сегментація медичного зображення. Після завершення сегментації поряд з початковим зображенням повинна відображатися структура в якій зберігатись данні сегментації (використовуючи різні кольори)

|  |  |
| --- | --- |
| **Номер варіанту** | **Алгоритм сегментації медичного об’єкту** |
| 3 | Метод нарощування ділянок |

**Лістинг програми**

**public class Main {**

public static void main(String[] args) {

//Deploying BufferedImage from DICOM to managers.DataManager...

DataManager.getInstance().setImage(ImageHelper.readDicom(DataManager.PATH));

launchSwingWithOpenGL2(DataManager.getInstance().getImage().getWidth(), DataManager.getInstance().getImage().getHeight());

}

public static void launchSwingWithOpenGL2(int width, int height) {

GLCanvas canvas = DataManager.getInstance().getCanvas();

canvas.setPreferredSize(new Dimension(width, height));

canvas.addGLEventListener(new Rendering());

canvas.addMouseListener(new MouseEventListener());

final JFrame frame = DataManager.getInstance().getFrame();

//Set the GUI content

frame.getContentPane().add(canvas, BorderLayout.CENTER);

//Set default options ...

frame.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

frame.pack();

frame.setLocationRelativeTo(null);

SwingUtilities.invokeLater( () -> frame.setVisible(true) );

}

}

**public class Pixel {**

public int color;

public int x;

public int y;

public Pixel(int grayScale) {

this.color = grayScale;

}

public Pixel(int grayScale, int x, int y){

this.color = grayScale;

this.x = x;

this.y = y;

}

@Override

public boolean equals(Object pixel){

if(pixel instanceof Pixel)

return this.x == ((Pixel) pixel).x && this.y == ((Pixel) pixel).y;

else

return false;

}

@Override

public String toString() {

return "[ ("+x+"|"+y+") grayScale = " + color + " ] ";

}

public void setColor(int grayScale) {

this.color = grayScale;

}

public int getGrayScale(){

return color;

}

public int getARGB(){

return new Color(color,color,color).getRGB();

}

public int getColoredARGB(){

return new Color(color,0,color).getRGB();

}

}

**public class Mouse {**

public int x;

public int y;

public void setPosition(int x, int y){

this.x = x;

this.y =y;

}

@Override

public String toString(){

return "Mouse ("+x+"|"+y+")";

}

}

**public class DataManager {**

private DataManager(){}

private static DataManager instance = null;

public static final String PATH = "lab7.dcm";

public static final int INTENSITY\_VARIATION = 60;

public static int NEW\_MAX\_GREYSCALE = 255;

private GLCanvas canvas = new GLCanvas();

private final JFrame frame = new JFrame();

private DICOM image=null;

private Mouse mouse = null;

public Mouse getMouse() {

if(mouse==null)

mouse = new Mouse();

return mouse;

}

public boolean isMousedClicked(){

return mouse == null;

}

public JFrame getFrame() {

return frame;

}

public GLCanvas getCanvas() {

return canvas;

}

public DICOM getImage() {

return image;

}

public void setImage(DICOM image) {

this.image = image;

}

public static DataManager getInstance(){

if(instance == null)

instance = new DataManager();

return instance;

}

}

**public class Rendering implements GLEventListener {**

@Override

public void init(GLAutoDrawable glAutoDrawable) {

GL2 gl2 = glAutoDrawable.getGL().getGL2();

gl2.glEnable(GL.GL\_TEXTURE\_2D);

}

@Override

public void reshape(GLAutoDrawable glAutoDrawable, int x, int y, int width, int height) {

GL2 gl2 = glAutoDrawable.getGL().getGL2();

gl2.glOrtho(0, width, 0, height, -1, 1);

}

@Override

public void display(GLAutoDrawable glAutoDrawable) {

GL2 gl2 = glAutoDrawable.getGL().getGL2();

gl2.glClear(GL.GL\_COLOR\_BUFFER\_BIT);

Texture currentTexture = null;

if(DataManager.getInstance().isMousedClicked())

currentTexture = AWTTextureIO.newTexture(glAutoDrawable.getGLProfile(), DataManager.getInstance().getImage().getBufferedImage(), false);

else

currentTexture = AWTTextureIO.newTexture(glAutoDrawable.getGLProfile(),

ImageHelper.getSegmentationImage(DataManager.getInstance().getImage(),DataManager.getInstance().getMouse()),

false);

currentTexture.bind(gl2);

BufferedImage image = DataManager.getInstance().getImage().getBufferedImage();

gl2.glBegin(GL2.GL\_POLYGON);

gl2.glTexCoord2i(0, 1);

gl2.glVertex2i(0, 0);

gl2.glTexCoord2i(0, 0);

gl2.glVertex2i(0, image.getHeight());

gl2.glTexCoord2i(1, 0);

gl2.glVertex2i(image.getWidth(), image.getHeight());

gl2.glTexCoord2i(1, 1);

gl2.glVertex2i(image.getWidth(), 0);

gl2.glEnd();

}

@Override

public void dispose(GLAutoDrawable glAutoDrawable) {

}

}

**public class ImageHelper {**

public static DICOM readDicom(String path) {

DICOM image = new DICOM();

image.open(path);

return image;

}

public static int getMax(List<Pixel> imageData){

return Collections.max(imageData,new Comparator<Pixel>() {

@Override

public int compare(Pixel o1, Pixel o2) {

return Integer.compare(o1.getGrayScale(),o2.getGrayScale());

}

}).getGrayScale();

}

public static int getMin(List<Pixel> imageData){

return Collections.min(imageData,new Comparator<Pixel>() {

@Override

public int compare(Pixel o1, Pixel o2) {

return Integer.compare(o1.getGrayScale(),o2.getGrayScale());

}

}).getGrayScale();

}

public static List<Pixel> createPixelsListFromImage(DICOM image){

List<Pixel> imageData = new ArrayList<>(image.getWidth()\*image.getHeight());

for(int i=0;i<image.getHeight();i++)

for(int j=0;j<image.getWidth();j++)

imageData.add(new Pixel(image.getPixel(i,j)[0]));

int minGrayScale = getMin(imageData);

List<Pixel> resultSet = new ArrayList<>(image.getWidth()\*image.getHeight());

for(Pixel pixel:imageData)

resultSet.add(new Pixel(pixel.getGrayScale() - minGrayScale));

return resultSet.isEmpty() ? null : resultSet;

}

public static Pixel[][] make2DimsMatrix(List<Pixel> pixels, int width, int height){

Pixel[][] pixelsMatrix = new Pixel[height][width];

for(int i=0;i<height;i++)

for(int j=0;j<width;j++)

pixelsMatrix[i][j] = pixels.get(i\*width + j);

return pixelsMatrix;

}

public static BufferedImage getSegmentationImage(DICOM origin, Mouse segmentationPoint) {

List<Pixel> pixelsList = createPixelsListFromImage(origin);

Pixel[][] pixels = make2DimsMatrix(pixelsList, origin.getWidth(), origin.getHeight());

List<Pixel> toCheck = new ArrayList<>();

List<Pixel> segmentationPoints = new ArrayList<>();

Pixel startPoint = new Pixel(pixels[segmentationPoint.x][segmentationPoint.y].getGrayScale(), segmentationPoint.x, segmentationPoint.y);

toCheck.add(startPoint);

while (!toCheck.isEmpty()) {

Pixel currentPoint = toCheck.get(0);

List<Pixel> neighbors = getNeighbors(pixels, currentPoint);

segmentationPoints.add(currentPoint);

toCheck.remove(currentPoint);

for (Pixel neighbor : neighbors)

if (isFitted(startPoint, neighbor) && !segmentationPoints.contains(neighbor) && !toCheck.contains(neighbor))

toCheck.add(neighbor);

}

double min = getMin(pixelsList);

double max = getMax(pixelsList);

pixelsList.forEach((pixel) -> pixel.setColor((int) ((pixel.getGrayScale() - min) / max \* DataManager.NEW\_MAX\_GREYSCALE)));

double minS = getMin(segmentationPoints);

double maxS = getMax(segmentationPoints);

segmentationPoints.forEach((pixel) -> pixel.setColor((int) ((pixel.getGrayScale() - minS) / maxS \* (DataManager.NEW\_MAX\_GREYSCALE/2) + DataManager.NEW\_MAX\_GREYSCALE/2)));

return makeColorImage(pixelsList, segmentationPoints, origin.getWidth(), origin.getHeight());

}

public static List<Pixel> getNeighbors(Pixel[][] allPixels, Pixel currentPixel){

List<Pixel> neighbors = new ArrayList<>(8);

for(int i=-1;i<2;i++){

for(int j=-1;j<2;j++){

if(i==0 &&j==0)

continue;

int x = currentPixel.x + j;

int y = currentPixel.y + i;

if(x>=0 && y>=0 && x<256 && y<256)

neighbors.add(new Pixel(allPixels[x][y].getGrayScale(),x,y));

}

}

return neighbors;

}

public static boolean isFitted(Pixel startPoint, Pixel currentPoint){

return startPoint.getGrayScale()- DataManager.INTENSITY\_VARIATION <= currentPoint.getGrayScale()

&& currentPoint.getGrayScale() <= startPoint.getGrayScale() + DataManager.INTENSITY\_VARIATION;

}

public static BufferedImage makeColorImage(List<Pixel> imagePixel, List<Pixel> coloredPixels, int width, int height){

BufferedImage result = new BufferedImage(width,height,BufferedImage.TYPE\_INT\_ARGB);

for(int i=0;i<height;i++){

for(int j=0;j<width;j++){

int indexOfColorPixel = coloredPixels.indexOf(new Pixel(0,i,j));

if(indexOfColorPixel!=-1){

result.setRGB(i,j, coloredPixels.get(indexOfColorPixel).getColoredARGB());

} else {

result.setRGB(i,j, imagePixel.get(i\*width+j).getARGB());

}

}

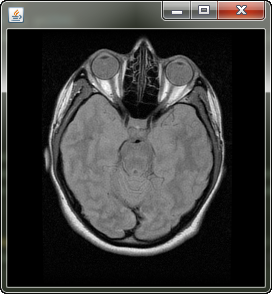
}

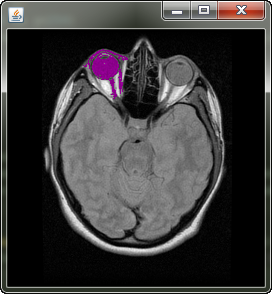
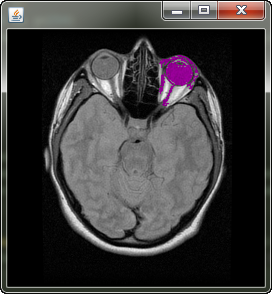
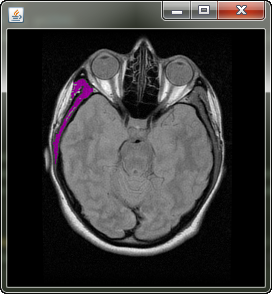
return result;

}

}

**Результати роботи програми**



|  |  |
| --- | --- |
|  |  |
|  |  |

**Відповіді на контрольні запитання**

**Відповіді на контрольні запитання:**

1. **Основна мета методів сегментації, заснованих на формуванні однорідних ділянок.**

Даний метод використовують з метою забезпечення максимальної якості визначення границь анатомічної структури ділянок, які цікавлять (пухлини, гематоми, тромби і т.п.).

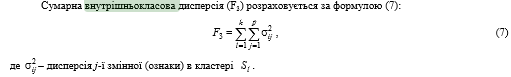
Методи,засновані на формуванні однорідних ділянок, починаються або від неоднорідності на границях, або від однорідності всередині ділянки. У цій групі методів можна виділити дві підгрупи: без врахування і з врахуванням просторових зв’язків. До методів бе зврахування просторових зв’язків належать метод порогової фільтрації і метод кластеризації. Оригінальні КТ-зображення є сірошкальними (єдиною характерною ознакою пік селів таких зображень може бути тільки яскравість). Можуть мати значення яскравості, що змінюються в рамках всього доступного діапазону градацій сірого. Таким чином, як знаходження порога для відсіювання яскравостей пікселів на гістограмах, такі введення міри близькості для визначення метрики на просторі ознак є неможливими, а це є ключовим завданням при реалізації методів сегментації без врахування просторових зв’язків. Отже, використання цих методів неможливе. До методів із врахуванням просторових зв’язків належать метод нарощування ділянок і метод розбиття-злиття ділянок. При використанні цих методів з метою сегментації зображень погано виділяються дрібні об’єкти. Це відбувається через те, що строго однорідні за характерною ознакою ділянки містять багато малих отворів та мають зубчасті границі, а вимога істотної різниці суміжних ділянок за значенням характерної ознаки призводить до злиття кількох ділянок.

1. **Суть сегментації методом порогової фільтрації.**

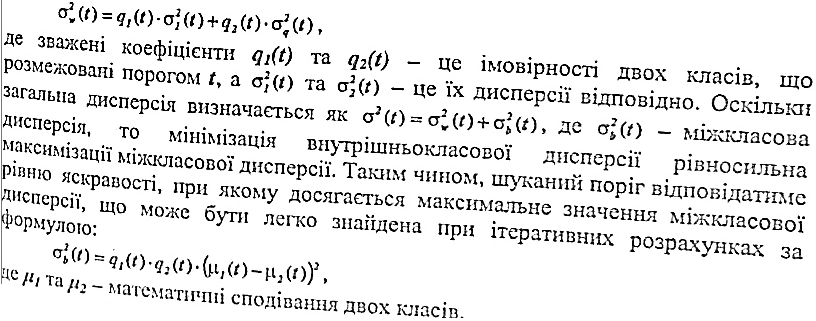
При використанні цього методу будуються гістограми характерних ознак, при цьому значення ознаки кожного піксела прирівнюється з деяким порогом гістограм, і в залежності від результату порівняння цей піксель відноситься до 1 з ділянок зображення.

1. **Внутрішньокласова дисперсія, її визначення.**

Внутрішньокласова дисперсія – це зважена сума дисперсій класів; функціонал якості розподілу об’єктів, для відбору найбільш якісної класифікації.



Чим менше значення внутрішньокласової дисперсії тим якісніша класифікація.



1. **Етапи реалізації алгоритму Оцу для сегментації методом порогової фільтрації.**

Швидким та ефективним методом обрання порога при роботі з гістограмою яскравостей зображення є метод Оцу. В цьому методі поріг фільтрації визначається за допомогою ітераційного алгоритму, в результаті роботи якого отримуваний поріг зменшує внутрішньокласову дисперсію, що встановлюється як зважена сума дисперсій двох класів.

Алгоритм сегментації за допомогою порогової фільтрації складається з трьох етапів:

1. Розрахунок значень гістограми для кожного рівня яскравості зображення
2. Визначення порогу для фільтрації (після проведення ітеративних розрахунків, шуканий поріг відповідатиме рівню яскравості, для якого міжкласова дисперсія набула максимального значення)
3. Проведення бінаризації зображення (розбиття зображення на 2 класи проводиться відносно знайденого порога)
4. **Основний принцип сегментації з використанням кластерного аналізу.**

В цьому методі завдання сегментації зводиться до завдання кластеризації завдяки відображенню пікселів зображення в деякому просторі ознак та введенні метрики на цьому просторі, що дає змогу використати для сегментації будь-який з методів кластерного аналізу. Поширеним методом с сегментація з використанням кластеризації методом k-середніх.

1. **Встановлення належності піксела зображення до одного з кластерів при використанні кластеризації методом k-середніх.**

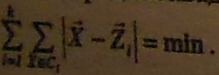
Алгоритм складається з 3-ох етапів:

1 – Ініціалізація (обирається k пік селів зображення, що стають початковими значеннями центра мас кластерів).

2 – Ітеративний пошук k середніх (проводиться розподіл пік селів на k класів, враховуючи наближення до центра мас кластерів, після чого проводять перерахунок центрів мас, усереднюючи піксели в кластерах відповідно до поточного розподілу).

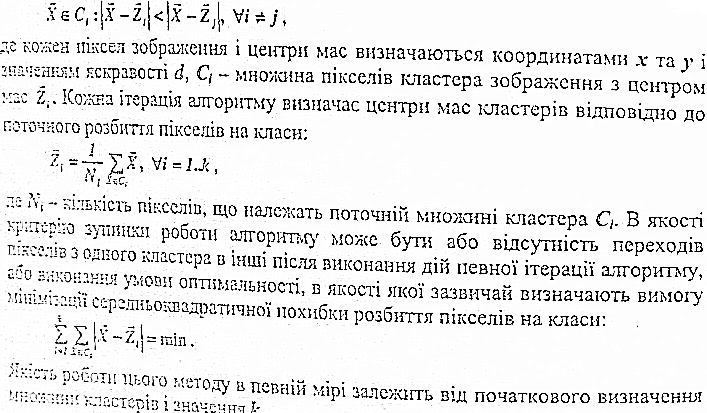
3 – Зупинка роботи алгоритму (після досягнення критерію зупинки отримані класи пік селів стають шуканими сегментами зображення).

1. **В якості критерію зупинки роботи алгоритму може бути або відсутність** переходів пік селів з одного кластеру в інші після виконання дій певної ітерації алгоритму, або виконання умови оптимальності, в якості якої зазвичай визначають вимогу мінімізації середньоквадратичної похибки розбиття пік селів на класи:



Якість роботи цього методу в певній мірі залежить від початкового визначення множини кластерів і значення k.

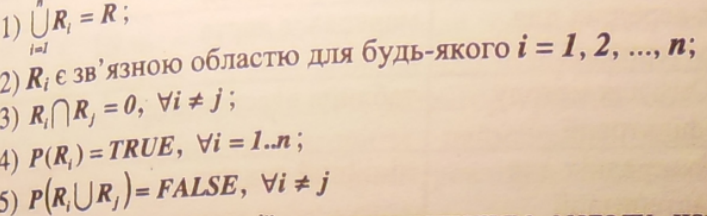
8) Алгоритм цього методу ітеративно визначає значення для центрів мас кожного з кластерів і розбиває піксели зображення на класи відповідно до визначених центрів мас. Належність піксела зображення Х(х,у,d) до одного з кластерів встановлюється через визначення найближчого до піксела центра мас Z(х,у,d):



1. **Основний принцип сегментації методом нарощування ділянок.**

Після інтерактивного обрання початкової точки, що належить певній ділянці зображення, алгоритми цього методу, так би мовити, «вирощують відповідну ділянку» аж до її границі, використовуючи критерій однорідності сусідніх пікселів.

При цьому сегменти R1, R2,…,R3 зображення R, що відповідають однорідним ділянкам за певним критерієм Р, мають володіти наступними властивостями:



1. Алгоритм сегментації методом нарощування ділянок складається з 2-ох етапів:
2. Ініціалізація (обираються початкові піксели на зображенні, що однозначно належать різним ділянкам, та встановлюється критерій однорідності)
3. Визначення сегментів (відбувається ітеративне об’єднання початкових пік селів із сусідніми доки виконуються правила відповідно до заданого критерію однорідності)
4. +13)

