Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп’ютерних наук

Кафедра безпеки інформаційних систем і технологій

Лабораторна робота №4

*з навчальної дисципліни*

«Стеганографія»

Виконав:

Студент групи КБ-41

Кравченко Є.М.

Перевірив:

доцент

Нарєжній О.П.

Харків – 2020 р.

**Тема:** приховування даних у частотній області нерухомих зображень на основі кодування різниці абсолютних значень коефіцієнтів дискретнокосинусного перетворення стеганографічних систем, дослідити властивості стеганографічних методів, що засновані на низькорівневих властивостях зорової системи людини (ЗСЛ).

**Мета роботи:** закріпити теоретичні знання з теми «Приховування даних у частотній області нерухомих зображень», набути практичних вмінь та навичок з розробки стеганографічних систем, дослідити властивості стеганографічних методів, заснованих на низькорівневих властивостях зорової системи людини (ЗСЛ), зокрема частотної чутливості.

**1. Реалізація алгоритмів прямого та зворотного дискретно-косинусного перетворення. Дослідження ефекту частотної чутливості зорової системи людини**

Реалізація алгоритмів прямого дискретно-косинусного перетворення:

**protected double**[][] forwardDCT(**double**[][] input) {  
 **double**[][] output = **new double**[**N**][**N**];  
 **for** (**int** i = 0; i < **N**; i++) {  
 **for** (**int** j = 0; j < **N**; j++) {  
 **double** sum = 0d;  
 **for** (**int** x = 0; x < **N**; x++) {  
 **for** (**int** y = 0; y < **N**; y++) {  
 sum += input[x][y] \* Math.*cos*(((2 \* x + 1) / (2.0 \* **N**)) \* i \* Math.***PI***) \* Math.*cos*(((2 \* y + 1) / (2.0 \* **N**)) \* j \* Math.***PI***);  
 }  
 }  
 sum \*= **c**[i] \* **c**[j] / Math.*sqrt*(2 \* **N**);  
 output[i][j] = sum;  
 }  
 }  
 **return** output;  
}

Реалізація алгоритмів зворотногодискретно-косинусного перетворення:

**protected double**[][] inverseDCT(**double**[][] input) {  
 **double**[][] output = **new double**[**N**][**N**];  
 **for** (**int** x = 0; x < **N**; x++) {  
 **for** (**int** y = 0; y < **N**; y++) {  
 **double** sum = 0d;  
 **for** (**int** i = 0; i < **N**; i++) {  
 **for** (**int** j = 0; j < **N**; j++) {  
 sum += **c**[i] \* **c**[j] \* input[i][j] \* Math.*cos*(((2 \* x + 1) / (2.0 \* **N**)) \* i \* Math.***PI***) \* Math.*cos*(((2 \* y + 1) / (2.0 \* **N**)) \* j \* Math.***PI***);  
 }  
 }  
 sum /= Math.*sqrt*(2 \* **N**);  
 output[x][y] = sum;  
 }  
 }  
 **return** output;  
}

**2. Реалізація алгоритмів вбудовування та вилучення повідомлень до частотної області зображень (метод Коха–Жао)**

Алгоритм вбудування:

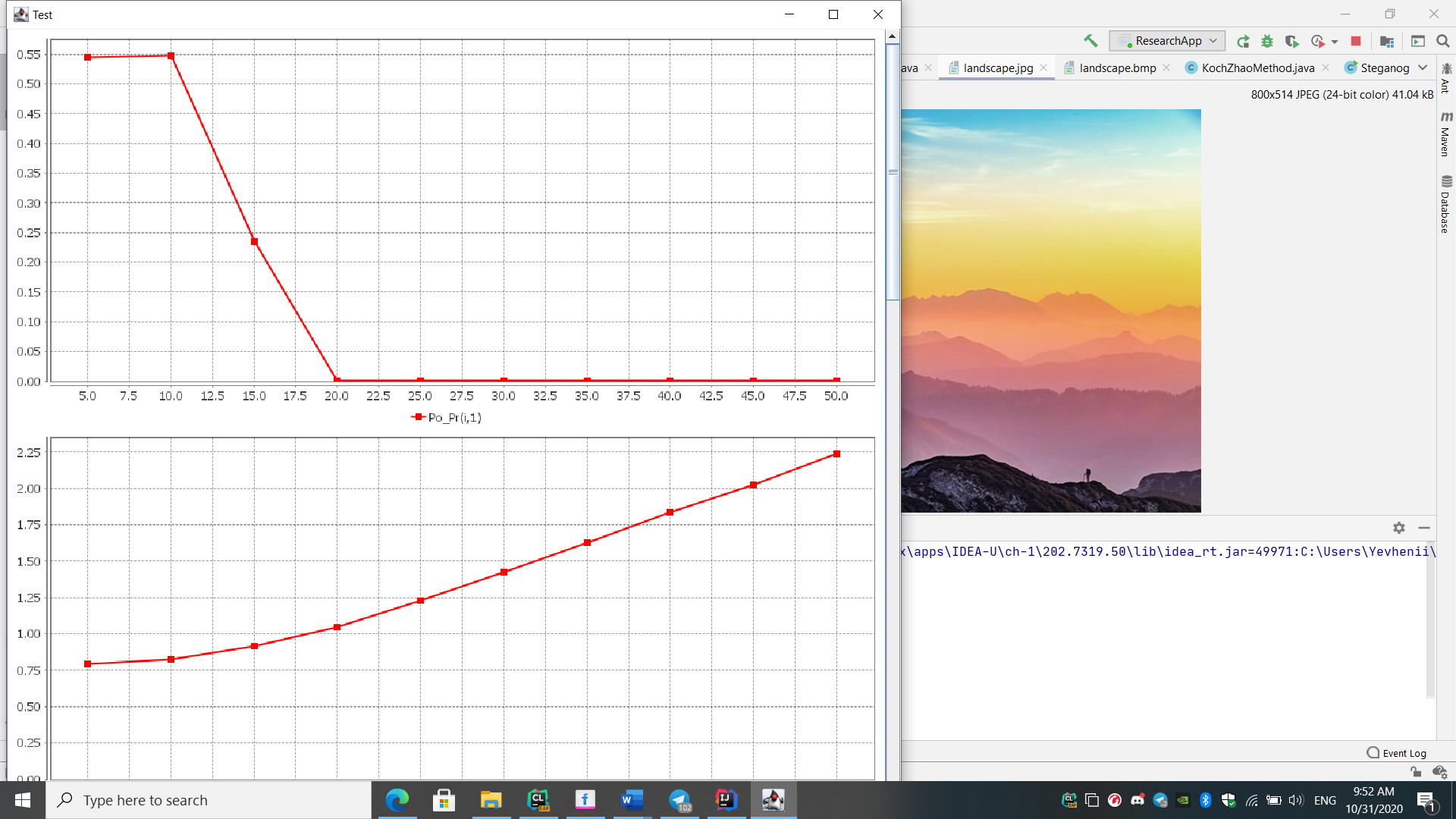
@Override  
**public** BufferedImage encode(BufferedImage bufferedImage, String message) **throws** ThroughputException {  
 **int** height = (bufferedImage.getHeight() / **N**) \* **N**;  
 **int** width = (bufferedImage.getWidth() / **N**) \* **N**;  
 **double**[][] Y = **new double**[height][width];  
 **double**[][] Cb = **new double**[height][width];  
 **double**[][] Cr = **new double**[height][width];  
 **for** (**int** i = 0; i < height; i++) {  
 **for** (**int** j = 0; j < width; j++) {  
 Color color = **new** Color(bufferedImage.getRGB(j, i));  
 Y[i][j] = (**int**) (0 + (0.299 \* color.getRed()) + (0.587 \* color.getGreen()) + (0.114 \* color.getBlue()));  
 Cb[i][j] = (**int**) (128 - (0.168736 \* color.getRed()) - (0.331264 \* color.getGreen()) + (0.50000 \* color.getBlue()));  
 Cr[i][j] = (**int**) (128 + (0.50000 \* color.getRed()) - (0.418688 \* color.getGreen()) - (0.081312 \* color.getBlue()));  
 }  
 }  
 ArrayList<**double**[][]> dctBlocks = **new** ArrayList<>();  
 DCT dct = **new** DCT(**N**);  
 **for** (**int** i = 0; i < height; i += **N**) {  
 **for** (**int** j = 0; j < width; j += **N**) {  
 **double**[][] tmp = **new double**[**N**][**N**];  
 **for** (**int** k = i, x = 0; k < i + **N**; k++, x++) {  
 **for** (**int** l = j, y = 0; l < j + **N**; l++, y++) {  
 tmp[x][y] = Y[k][l];  
 }  
 }  
 dctBlocks.add(dct.forwardDCT(tmp).clone());  
 }  
 }  
 String mb = toBinaryString(message);  
 **for** (**int** i = 0; i < Math.*min*(dctBlocks.size(), mb.length()); i++) {  
 **int** i2 = **N** / 3;  
 **int** i1 = i2 + i2;  
 **if** ((mb.charAt(i) == **'1'**)) {  
 **if** (dctBlocks.get(i)[i1][i2] > 0) {  
 dctBlocks.get(i)[i1][i2] = Math.*abs*(dctBlocks.get(i)[i2][i1]) + **Pr**;  
 } **else** {  
 dctBlocks.get(i)[i1][i2] = -Math.*abs*(dctBlocks.get(i)[i2][i1]) - **Pr**;  
 }  
 } **else** {  
 **if** (dctBlocks.get(i)[i2][i1] > 0) {  
 dctBlocks.get(i)[i2][i1] = Math.*abs*(dctBlocks.get(i)[i1][i2]) + **Pr**;  
 } **else** {  
 dctBlocks.get(i)[i2][i1] = -Math.*abs*(dctBlocks.get(i)[i1][i2]) - **Pr**;  
 }  
 }  
 }  
 **int** index = 0;  
 **for** (**int** i = 0; i < height; i += **N**) {  
 **for** (**int** j = 0; j < width; j += **N**) {  
 **double**[][] tmp = dct.inverseDCT(dctBlocks.get(index));  
 **for** (**int** k = i, x = 0; k < i + **N**; k++, x++) {  
 **for** (**int** l = j, y = 0; l < j + **N**; l++, y++) {  
 Y[k][l] = tmp[x][y];  
 }  
 }  
 index++;  
 }  
 }  
  
 **for** (**int** i = 0; i < height; i++) {  
 **for** (**int** j = 0; j < width; j++) {  
 **int** r = (**int**) (Y[i][j] + 1.40200 \* (Cr[i][j] - 0x80));  
 **int** g = (**int**) (Y[i][j] - 0.34414 \* (Cb[i][j] - 0x80) - 0.71414 \* (Cr[i][j] - 0x80));  
 **int** b = (**int**) (Y[i][j] + 1.77200 \* (Cb[i][j] - 0x80));  
 r = Math.*max*(0, Math.*min*(255, r));  
 g = Math.*max*(0, Math.*min*(255, g));  
 b = Math.*max*(0, Math.*min*(255, b));  
 bufferedImage.setRGB(j, i, **new** Color(r, g, b).getRGB());  
 }  
 }  
 **return** bufferedImage;  
}

Алгоритм вилучення:

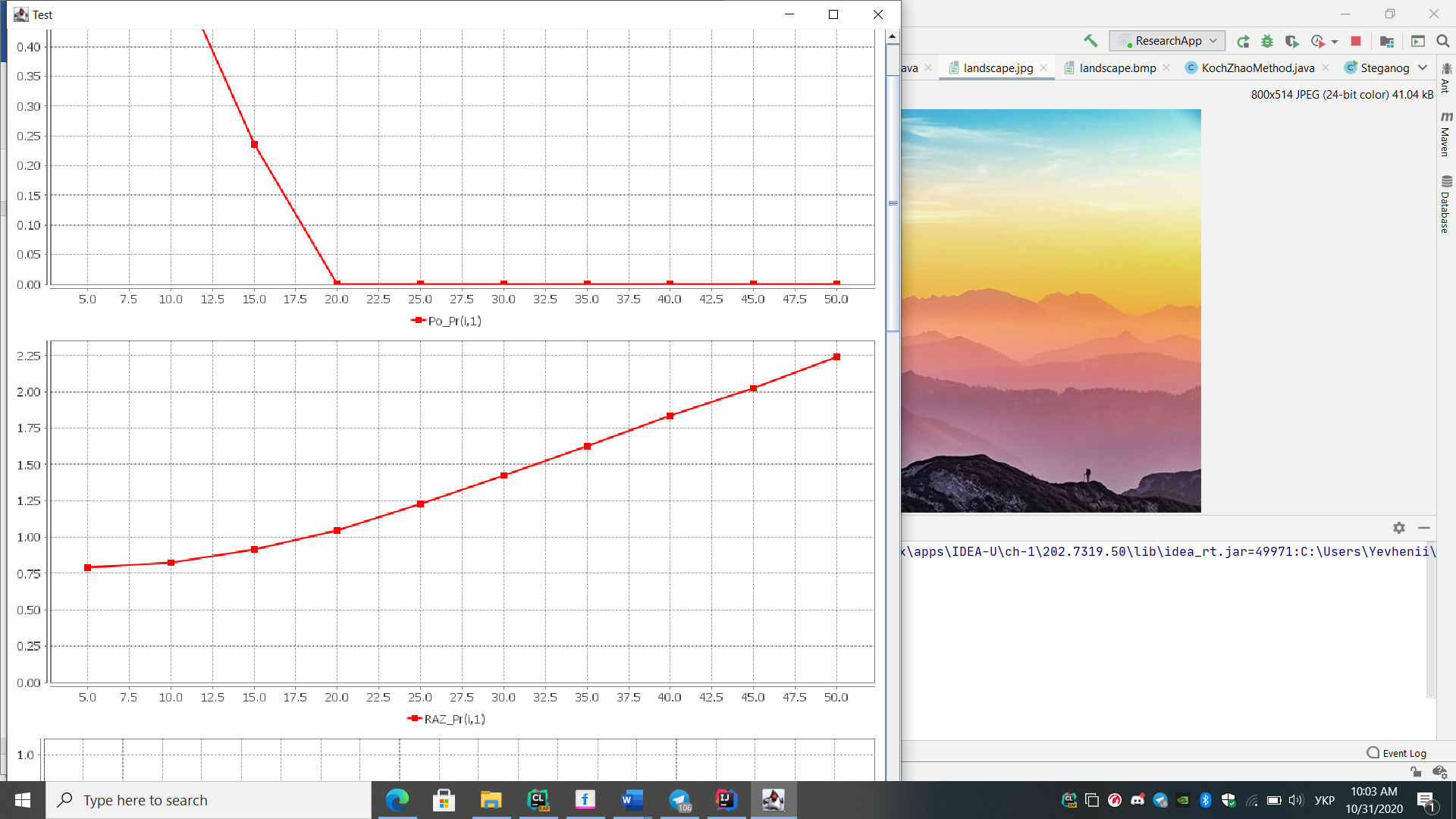
@Override  
**public** String decode(BufferedImage bufferedImage) {  
 StringBuilder mb = **new** StringBuilder();  
 **int** height = (bufferedImage.getHeight() / **N**) \* **N**;  
 **int** width = (bufferedImage.getWidth() / **N**) \* **N**;  
 **double**[][] Y = **new double**[height][width];  
 **double**[][] Cb = **new double**[height][width];  
 **double**[][] Cr = **new double**[height][width];  
 **for** (**int** i = 0; i < height; i++) {  
 **for** (**int** j = 0; j < width; j++) {  
 Color color = **new** Color(bufferedImage.getRGB(j, i));  
 Y[i][j] = (**int**) (0 + (0.299 \* color.getRed()) + (0.587 \* color.getGreen()) + (0.114 \* color.getBlue()));  
 Cb[i][j] = (**int**) (128 - (0.168736 \* color.getRed()) - (0.331264 \* color.getGreen()) + (0.50000 \* color.getBlue()));  
 Cr[i][j] = (**int**) (128 + (0.50000 \* color.getRed()) - (0.418688 \* color.getGreen()) - (0.081312 \* color.getBlue()));  
  
 }  
 }  
 ArrayList<**double**[][]> dctImage = **new** ArrayList<>();  
 DCT dct = **new** DCT(**N**);  
 **for** (**int** i = 0; i < height; i += **N**) {  
 **for** (**int** j = 0; j < width; j += **N**) {  
 **double**[][] tmp = **new double**[**N**][**N**];  
 **for** (**int** k = i, x = 0; k < i + **N**; k++, x++) {  
 **for** (**int** l = j, y = 0; l < j + **N**; l++, y++) {  
 tmp[x][y] = Y[k][l];  
 }  
 }  
 dctImage.add(dct.forwardDCT(tmp.clone()));  
 }  
 }  
 **for** (**double**[][] doubles : dctImage) {  
 **int** i1 = **N** / 3;  
 **int** i = i1 + i1;  
 **if** (Math.*abs*(doubles[i][i1]) > Math.*abs*(doubles[i1][i])) {  
 mb.append(**'1'**);  
 } **else** {  
 mb.append(**'0'**);  
 }  
 }  
 **byte**[] result = **new byte**[mb.length() / 8];  
 **for** (**int** i = 0, k = 0; i + 8 < mb.length(); i += 8) {  
 **byte** b = (**byte**) (**int**) Integer.*valueOf*(mb.substring(i, i + 8), 2);  
 result[k++] = b;  
 }  
 **return new** String(result, StandardCharsets.***UTF\_8***);  
}

**3. Реалізація стеганоатаки на основі використання алгоритму стискання JPEG та дослідження її можливостей**

Графік залежності ймовірності помилкового вилучення від коефіцієнта “Pr”:



Графік залежності спотворення контейнера від коефіцієнта “Pr”:



**4. Реалізація вдосконалених алгоритмів вбудовування та вилучення повідомлень до частотної області зображень (метод Бенгама–Мемона–Ео–Юнг)**

Алгоритм вбудування:

@Override  
**public** BufferedImage encode(BufferedImage bufferedImage, String message) **throws** ThroughputException {  
 **int** height = (bufferedImage.getHeight() / **N**) \* **N**;  
 **int** width = (bufferedImage.getWidth() / **N**) \* **N**;  
 **double**[][] Y = **new double**[height][width];  
 **double**[][] Cb = **new double**[height][width];  
 **double**[][] Cr = **new double**[height][width];  
 **for** (**int** i = 0; i < height; i++) {  
 **for** (**int** j = 0; j < width; j++) {  
 Color color = **new** Color(bufferedImage.getRGB(j, i));  
 Y[i][j] = (**int**) (0 + (0.299 \* color.getRed()) + (0.587 \* color.getGreen()) + (0.114 \* color.getBlue()));  
 Cb[i][j] = (**int**) (128 - (0.168736 \* color.getRed()) - (0.331264 \* color.getGreen()) + (0.50000 \* color.getBlue()));  
 Cr[i][j] = (**int**) (128 + (0.50000 \* color.getRed()) - (0.418688 \* color.getGreen()) - (0.081312 \* color.getBlue()));  
 }  
 }  
 ArrayList<**double**[][]> dctBlocks = **new** ArrayList<>();  
 DCT dct = **new** DCT(**N**);  
 **for** (**int** i = 0; i < height; i += **N**) {  
 **for** (**int** j = 0; j < width; j += **N**) {  
 **double**[][] tmp = **new double**[**N**][**N**];  
 **for** (**int** k = i, x = 0; k < i + **N**; k++, x++) {  
 **for** (**int** l = j, y = 0; l < j + **N**; l++, y++) {  
 tmp[x][y] = Y[k][l];  
 }  
 }  
 dctBlocks.add(dct.forwardDCT(tmp).clone());  
 }  
 }  
 String mb = toBinaryString(message);  
 **int** ind2 = **N** / 3;  
 **int** ind1 = ind2 + ind2;  
 **int** ind3 = (ind1 + ind2) / 2;  
 **for** (**int** i = 0; i < Math.*min*(dctBlocks.size(), mb.length()); i++) {  
 **if** ((mb.charAt(i) == **'1'**)) {  
 **if** (dctBlocks.get(i)[ind3][ind3] > 0) {  
 dctBlocks.get(i)[ind3][ind3] = Math.*max*(Math.*abs*(dctBlocks.get(i)[ind2][ind1]), Math.*abs*(dctBlocks.get(i)[ind1][ind2])) + **Pr**;  
 } **else** {  
 dctBlocks.get(i)[ind3][ind3] = -Math.*max*(Math.*abs*(dctBlocks.get(i)[ind2][ind1]), Math.*abs*(dctBlocks.get(i)[ind1][ind2])) - **Pr**;  
 }  
 } **else** {  
 **if** (dctBlocks.get(i)[ind3][ind3] > 0) {  
 dctBlocks.get(i)[ind3][ind3] = Math.*min*(Math.*abs*(dctBlocks.get(i)[ind2][ind1]), Math.*abs*(dctBlocks.get(i)[ind1][ind2])) - **Pr**;  
 } **else** {  
 dctBlocks.get(i)[ind3][ind3] = -Math.*min*(Math.*abs*(dctBlocks.get(i)[ind2][ind1]), Math.*abs*(dctBlocks.get(i)[ind1][ind2])) + **Pr**;  
 }  
 }  
 }  
 **for** (**int** i = 0, index = 0; i < height; i += **N**) {  
 **for** (**int** j = 0; j < width; j += **N**, index++) {  
 **double**[][] tmp = dct.inverseDCT(dctBlocks.get(index));  
 **for** (**int** k = i, x = 0; k < i + **N**; k++, x++) {  
 **for** (**int** l = j, y = 0; l < j + **N**; l++, y++) {  
 Y[k][l] = tmp[x][y];  
 }  
 }  
 }  
 }  
 **for** (**int** i = 0; i < height; i++) {  
 **for** (**int** j = 0; j < width; j++) {  
 **int** r = (**int**) (Y[i][j] + 1.40200 \* (Cr[i][j] - 0x80));  
 **int** g = (**int**) (Y[i][j] - 0.34414 \* (Cb[i][j] - 0x80) - 0.71414 \* (Cr[i][j] - 0x80));  
 **int** b = (**int**) (Y[i][j] + 1.77200 \* (Cb[i][j] - 0x80));  
 r = Math.*max*(0, Math.*min*(255, r));  
 g = Math.*max*(0, Math.*min*(255, g));  
 b = Math.*max*(0, Math.*min*(255, b));  
 bufferedImage.setRGB(j, i, **new** Color(r, g, b).getRGB());  
 }  
 }  
 **return** bufferedImage;  
}

Алгоритм вилучення:

@Override  
**public** String decode(BufferedImage bufferedImage) {  
 StringBuilder mb = **new** StringBuilder();  
 **int** height = (bufferedImage.getHeight() / **N**) \* **N**;  
 **int** width = (bufferedImage.getWidth() / **N**) \* **N**;  
 **double**[][] Y = **new double**[height][width];  
 **double**[][] Cb = **new double**[height][width];  
 **double**[][] Cr = **new double**[height][width];  
 **for** (**int** i = 0; i < height; i++) {  
 **for** (**int** j = 0; j < width; j++) {  
 Color color = **new** Color(bufferedImage.getRGB(j, i));  
 Y[i][j] = (**int**) (0 + (0.299 \* color.getRed()) + (0.587 \* color.getGreen()) + (0.114 \* color.getBlue()));  
 Cb[i][j] = (**int**) (128 - (0.168736 \* color.getRed()) - (0.331264 \* color.getGreen()) + (0.50000 \* color.getBlue()));  
 Cr[i][j] = (**int**) (128 + (0.50000 \* color.getRed()) - (0.418688 \* color.getGreen()) - (0.081312 \* color.getBlue()));  
  
 }  
 }  
 ArrayList<**double**[][]> dctBlocks = **new** ArrayList<>();  
 DCT dct = **new** DCT(**N**);  
 **for** (**int** i = 0; i < height; i += **N**) {  
 **for** (**int** j = 0; j < width; j += **N**) {  
 **double**[][] tmp = **new double**[**N**][**N**];  
 **for** (**int** k = i, x = 0; k < i + **N**; k++, x++) {  
 **for** (**int** l = j, y = 0; l < j + **N**; l++, y++) {  
 tmp[x][y] = Y[k][l];  
 }  
 }  
 dctBlocks.add(dct.forwardDCT(tmp.clone()));  
 }  
 }  
 **int** ind2 = **N** / 3;  
 **int** ind1 = ind2 + ind2;  
 **int** ind3 = (ind1 + ind2) / 2;  
 **for** (**double**[][] doubles : dctBlocks) {  
 **if** (Math.*abs*(doubles[ind3][ind3]) > Math.*max*(Math.*abs*(doubles[ind2][ind1]), Math.*abs*(doubles[ind1][ind2]))) {  
 mb.append(**'1'**);  
 } **else** {  
 mb.append(**'0'**);  
 }  
 }  
 **byte**[] result = **new byte**[mb.length() / 8];  
 **for** (**int** i = 0, k = 0; i + 8 < mb.length(); i += 8) {  
 **byte** b = (**byte**) (**int**) Integer.*valueOf*(mb.substring(i, i + 8), 2);  
 result[k++] = b;  
 }  
 **return new** String(result, StandardCharsets.***UTF\_8***);  
}

**Висновок**

В цій лабораторній роботі було закріплено теоретичні знання за темою «Приховування даних у просторовій області нерухомих зображень на основі прямого розширення спектру», набуто практичних вмінь та навичок щодо розробки стеганографічних систем, досліджено властивості стеганографічних методів, що засновані на низькорівневих властивостях зорової системи людини (ЗСЛ).