Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп’ютерних наук

Кафедра безпеки інформаційних систем і технологій

Лабораторна робота №2

*з навчальної дисципліни*

«Стеганографія»

Виконав:

Студент групи КБ-41

Кравченко Є.М.

Перевірив:

доцент

Нарєжній О.П.

Харків – 2020 р.

**Тема:** приховування даних у просторовій області нерухомих зображень методом блокового вбудовування, методом квантування та методом «хреста».

**Мета роботи:** закріпити теоретичні знання за темою «Приховування даних у просторовій області нерухомих зображень методом блокового вбудовування, методом квантування та методом "хреста"», набути практичних вмінь та навичок щодо розробки стеганографічних систем, дослідити властивості стеганографічних методів, що засновані на низькорівневих властивостях зорової системи людини (ЗСЛ).

**1. Реалізація алгоритмів вбудовування та вилучення**

**повідомлень методом блокового вбудовування**

Алгоритм вбудовування даних:

@Override  
**public** BufferedImage encode(BufferedImage bufferedImage, String message) **throws** ThroughputException {  
 String mb = toBinaryString(message);  
 **if** (mb.length() > bufferedImage.getWidth()) {  
 **throw new** ThroughputException();  
 }  
 **for** (**int** i = 0; i < bufferedImage.getWidth() && i < mb.length(); i++) {  
 **int** b = 0;  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 b += **new** Color(bufferedImage.getRGB(i, j)).getRed();  
 }  
 b = b % 2;  
 Color color = **new** Color(bufferedImage.getRGB(i, 0));  
 **if** (b != charToInt(mb.charAt(i))) {  
 bufferedImage.setRGB(i, 0, **new** Color(color.getRed() + 1, color.getGreen(), color.getBlue()).getRGB());  
 }  
 }  
 **return** bufferedImage;  
}

Алгоритм вилучення даних:

@Override  
**public** String decode(BufferedImage bufferedImage) {  
 StringBuilder mb = **new** StringBuilder();  
 **for** (**int** i = 0; i < bufferedImage.getWidth(); i++) {  
 **int** b = 0;  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 b += **new** Color(bufferedImage.getRGB(i, j)).getRed();  
 }  
 b = b % 2;  
 mb.append(b);  
 }  
 **byte**[] result = **new byte**[bufferedImage.getWidth() / 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***);  
}

**2. Реалізація алгоритмів вбудовування та вилучення**

**повідомлень методом квантування**

Формування таблиці квантування:

**private** Map<Integer, Integer> **d** = **new** HashMap<>();  
  
**public** ImageQuantumMethod() {  
 **for** (**int** i = -255; i < 256; i++) {  
 **d**.put(i, Math.*abs*(i % 2));  
 }  
}

Алгоритм вбудовування даних:

@Override  
**public** BufferedImage encode(BufferedImage bufferedImage, String message) **throws** ThroughputException {  
 String mb = toBinaryString(message);  
 **if** (mb.length() > (bufferedImage.getWidth() \* bufferedImage.getHeight() / 2)) {  
 **throw new** ThroughputException();  
 }  
 **int** k = 0;  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 **for** (**int** i = 1; i < bufferedImage.getWidth() - 1; i += 2) {  
 Color currentColor = **new** Color(bufferedImage.getRGB(i, j));  
 Color previousColor = **new** Color(bufferedImage.getRGB(i - 1, j));  
 **int** red0 = previousColor.getRed();  
 **int** red1 = currentColor.getRed();  
 **while** (**d**.get(red1 - red0) != charToInt(mb.charAt(k))) {  
 **if** (red0 + 1 < 255) {  
 red0++;  
 **continue**;  
 } **else** {  
 red0--;  
 }  
 **if** (red1 + 1 < 255) {  
 red1++;  
 } **else** {  
 red1--;  
 }  
 }  
 bufferedImage.setRGB(i - 1, j, **new** Color(red0, previousColor.getGreen(), previousColor.getBlue()).getRGB());  
 bufferedImage.setRGB(i, j, **new** Color(red1, currentColor.getGreen(), currentColor.getBlue()).getRGB());  
 **if** (++k == mb.length()) **return** bufferedImage;  
 }  
 }  
 **return** bufferedImage;  
}

Алгоритм вилучення даних:

@Override  
**public** String decode(BufferedImage bufferedImage) {  
 StringBuilder mb = **new** StringBuilder();  
 **for** (**int** i = 0; i < bufferedImage.getHeight(); i++) {  
 **for** (**int** j = 1; j < bufferedImage.getWidth() - 1; j += 2) {  
 Color currentColor = **new** Color(bufferedImage.getRGB(j, i));  
 Color previousColor = **new** Color(bufferedImage.getRGB(j - 1, i));  
 **int** b = **d**.get(currentColor.getRed() - previousColor.getRed());  
 mb.append(b);  
 }  
 }  
 **byte**[] result = **new byte**[(bufferedImage.getWidth() / 2 \* bufferedImage.getHeight()) / 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 алгоритмів вбудовування та вилучення**

**повідомлень методом Куттера–Джордана–Боссена (методом «хреста»)**

Алгоритм вбудовування даних:

@Override  
**public** BufferedImage encode(BufferedImage bufferedImage, String message) {  
 String mb = toBinaryString(message);  
 **int** k = 0;  
 **for** (**int** j = **sizeOfPredictionArea**; j < bufferedImage.getHeight() - **sizeOfPredictionArea**; j++) {  
 **for** (**int** i = j; i < bufferedImage.getWidth() - **sizeOfPredictionArea**; i += **sizeOfPredictionArea**) {  
 **if** (k + 1 == mb.length()) {  
 **return** bufferedImage;  
 }  
 Color color = **new** Color(bufferedImage.getRGB(i, j));  
 **double** a = 0.29890 \* color.getRed() + 0.58662 \* color.getGreen() + 0.11448 \* color.getBlue();  
 **int** newBlue = (**int**) (color.getBlue() + (2 \* chatToInt(mb.charAt(k++)) - 1) \* a \* **energyBuiltInBit**);  
 Color newColor = **new** Color(color.getRed(), color.getGreen(), newBlue < 0 ? 0 : Math.*min*(newBlue, 255));  
 bufferedImage.setRGB(i, j, newColor.getRGB());  
 *//bufferedImage.setRGB(i, j, 0);* }  
 }  
 **return** bufferedImage;  
}

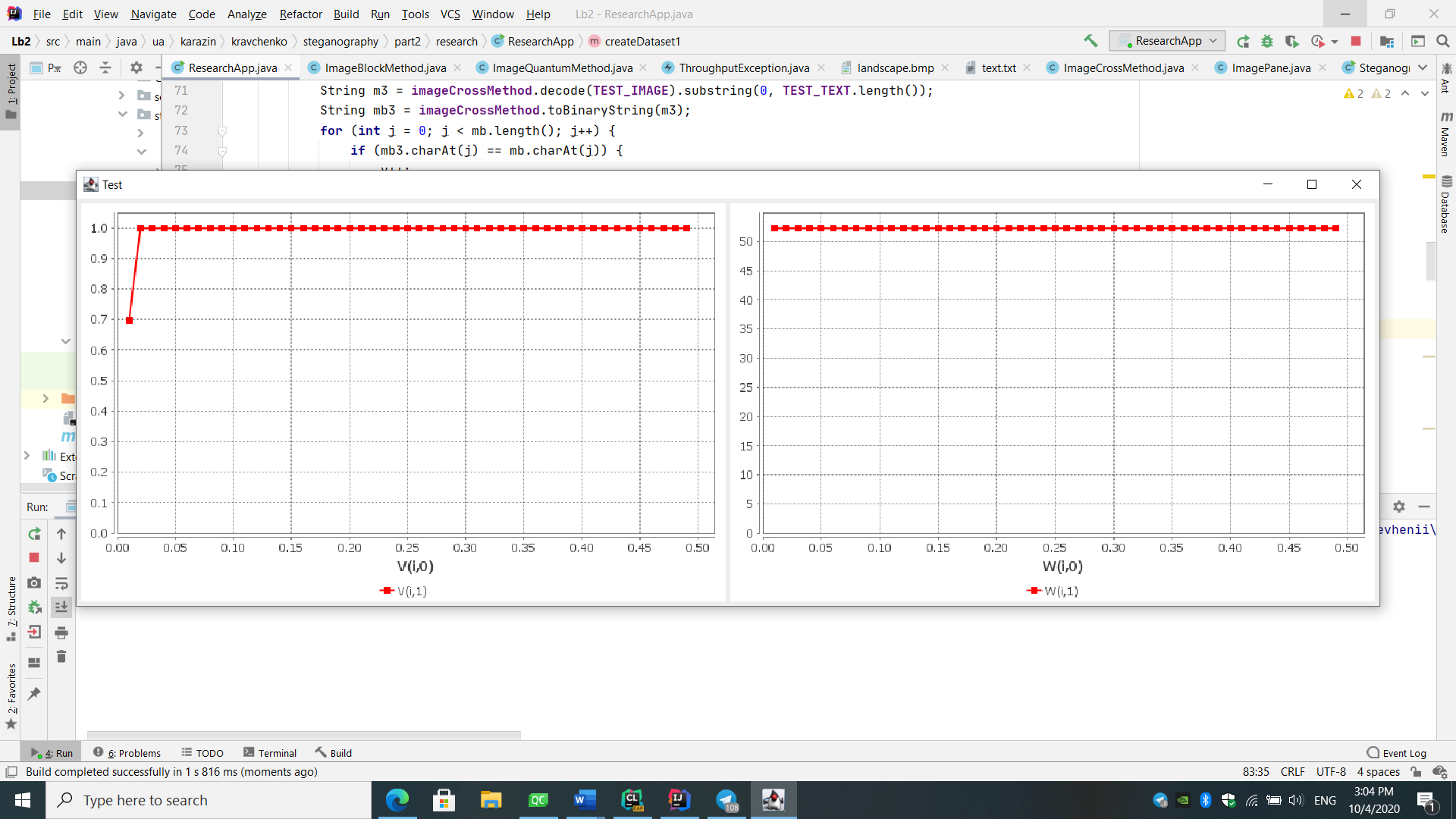
Алгоритм вилучення даних:

@Override  
**public** String decode(BufferedImage bufferedImage) {  
 StringBuilder mb = **new** StringBuilder();  
 **for** (**int** j = **sizeOfPredictionArea**; j < bufferedImage.getHeight() - **sizeOfPredictionArea**; j++) {  
 **for** (**int** i = j; i < bufferedImage.getWidth() - **sizeOfPredictionArea**; i += **sizeOfPredictionArea**) {  
 Color color = **new** Color(bufferedImage.getRGB(i, j));  
 **double** tmp = 0d;  
 **for** (**int** k = i - **sizeOfPredictionArea**; k <= i + **sizeOfPredictionArea**; k++) {  
 **if** (i != k) {  
 tmp += **new** Color(bufferedImage.getRGB(k, j)).getBlue();  
 }  
 }  
 **for** (**int** k = j - **sizeOfPredictionArea**; k <= j + **sizeOfPredictionArea**; k++) {  
 **if** (j != k) {  
 tmp += **new** Color(bufferedImage.getRGB(i, k)).getBlue();  
 }  
  
 }  
 tmp /= 4 \* **sizeOfPredictionArea**;  
 **double** b = color.getBlue() - tmp;  
 mb.append(b > 0 ? **'1'** : **'0'**);  
 }  
 }  
  
  
 **byte**[] result = **new byte**[bufferedImage.getWidth() / **sizeOfPredictionArea** \* (bufferedImage.getHeight() - 2 \* **sizeOfPredictionArea**) / 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***);  
  
}

**4. Дослідження ймовірнісних характеристикстеганографічного методу вбудовування данихКуттера–Джордана–Боссена (методу «хреста»)**

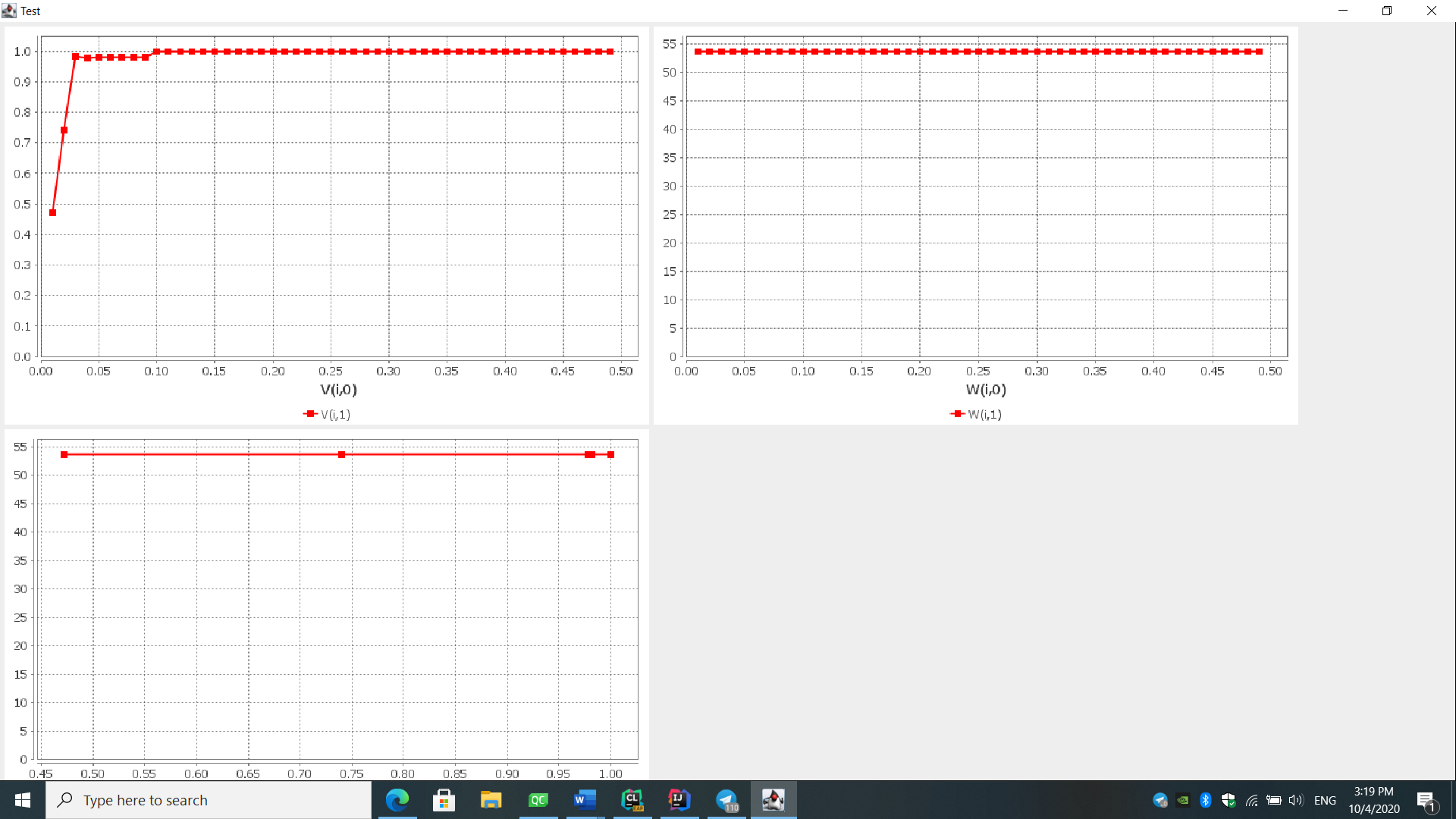
Проведемо оцінку ймовірності правильного вилучення повідомлення і величини внесених спотворень від коефіцієнта γ. Для цього будемо послідовно збільшувати величину γ і для кожного значення розраховувати частоту v правильно отриманих інформаційних бітів. Одночасно будемо розраховувати усереднену величину w внесених спотворень, виражену у відсотковому співвідношенні до максимального значення яскравості.

Побудуємо графіки отриманих емпіричних залежностей:



Зрозуміло, що величина внесених спотворень зростає лінійно від коефіцієнта γ. Однак емпірична залежність ймовірності правильного вилучення інформаційних даних поводиться інакше. При малих значеннях коефіцієнта γ величина W зростає швидко, однак при γ > 0,2 подальше збільшення енергії вбудовування не призводить до суттєвого підвищення ймовірності правильного вилучення, підвищувати величину V в даному випадку недоцільно.

Побудуємо інтегральний графік залежності величини W внесених спотворень у контейнер-зображення при забезпеченні відповідної ймовірності V правильного вилучення інформаційних даних:



Зрозуміло, що ефективне приховування вбудованих інформаційних даних без внесення значних спотворень (W <5 %) в контейнер-зображення буде спостерігатися тільки при ймовірності правильного вилучення даних V<0,8 ... 0,9, що відповідає енергії вбудовування γ = 0,05 ... 0,15. Підвищення достовірності отриманих даних за рахунок подальшого збільшення енергії вбудовування є недоцільним, оскільки це призводить до внесення невиправдано високих спотворень у контейнер-зображення. В даному випадку найбільш перспективною є реалізація завадостійкого кодування інформаційних даних і контроль помилок, що виникають при стеганографічних перетвореннях.

**Висновок**

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