ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ОЛЕСЯ ГОНЧАРА

Факультет прикладної математики
Кафедра обчислювальної математики та математичної кібернетики

КУРСОВА РОБОТА
за спеціальністю
на тему **«Наближені алгоритми розв'язання задачі реконструкції зображень»**

Другий (магістерський) рівень вищої освіти
Спеціальність 113 Прикладна математика
Освітня програма «Комп'ютерне моделювання та обчислювальні методи»

Виконавець
студентка групи ПМ-20м-1
Сіряк Аліна Юріївна

Керівник
професор кафедри обчислювальної
математики та математичної
кібернетики, д-р. фіз.-мат. наук _____Л.Л. Гарт

Кількість балів
Оцінка за національною шкалою _____

Члени комісії:

_____

_____

_____

2021

РЕФЕРАТ

Курсова робота: 36 с., 9 рис., 2 табл., 12 джерел, 1 додаток.

Об'єкт дослідження: наближені алгоритми розв’язання задачі реконструкції зображень.

Мета роботи: ознайомитися з існуючими алгоритмами відновлення зображень, реалізувати знайдені алгоритми на ЕОМ, дослідити якість та швидкість алгоритмів, зробити порівняльний аналіз та отримати висновки.

Одержані висновки та їх новизна: було вивчено та проаналізовано різні способи відновлення зображень, реалізовано деякі з відомих на сьогодення методів та детально описані алгоритми їх роботи. Була розглянута математична модель зображувальних систем, що описується набором одновимірних інтегральних рівнянь Фредгольма I роду типу згортки. Розроблено програмний продукт, що реалізує алгоритми методу регуляризації Тихонова та фільтрації Вінера, демонструє їх роботу на конкретному зображенні, а також представлено їх порівняльний аналіз.

Результати досліджень можуть бути застосовані при задачах компютерної графіки, у роботі з відео та фото, у медицинських досліженнях, у астрономії, у машинному навчанні та в задачах розпізнавання.

Перелік ключових слів: ОБРОБКА ЗОБРАЖЕНЬ, ФІЛЬТРИ ОБРОБКИ ЗОБРАЖЕНЬ, ЗГОРТКА, ФІЛЬТР ВІНЕРА, ІНТЕГРАЛЬНЕ РІВНЯННЯ І РОДУ, МЕТОД РЕГУЛЯРИЗАЦІЇ ТИХОНОВА, МЕТОД РЕГУЛЯРИЗАЦІЇ ФРІДМАНА, НАБЛИЖЕНИЙ РОЗВ’ЯЗОК.

ЗМІСТ

ВСТУП

Зображення об'єктів є важливою складовою частиною опису предметів і явищ навколишнього світу в самих передових галузях науки, техніки, медицини. Зображення часто буває спотворене внаслідок недосконалості оптичних апаратів, дефокусировки знімків, через вплив середовища між об'єктом і апаратом, в силу ряду інших причин.

В процесі передачі і перетворення за допомогою радіотехнічних систем, зображення піддаються впливу різних перешкод, що в ряді випадків призводить до погіршення візуального якості і втрати ділянок зображень. Широке використання комп'ютерної графіки в багатьох областях, таких як медицина, технічна діагностика, мультимедійні та освітні програми та ін., робить актуальними завдання обробки графічної інформації. На практиці часто зустрічаються зображення, спотворені шумом, який з'являється на етапах формування та передачі його по каналу зв'язку [5].

Відновлення спотворених зображень є однією з найбільш цікавих і важливих проблем в задачах обробки зображень - як з теоретичної, так і з практичної точок зору.

При спотворенні вся інформація перерозподіляється по деякому закону і може бути однозначно відновлена з деякими обмеженнями [4].

Розподіл основних видів обробки зображень:

- Корекція зображень

Корекція геометричних спотворень передбачає зміну за певними правилами координат функцій, не змінюючи значень функцій в кожній точці. Такі перетворення застосовуються, наприклад, при зміні масштабу зображень, усунення відомих геометричних аберацій, викликаних системами формування. Особливо широко дане перетворення використовується в аерофотозйомки.

- Поліпшення візуальної якості зображення

При обробці зображень часто здійснюють інтерактивну зміну значень функції зображення, тобто локально змінюють значення функції, не змінюючи координат (області завдання). Фактично це ретуш, яку часто застосовували і застосовують фотографи. Для перетворень зображень такого роду існують комплекси стандартних програм, якими оснащені персональні комп'ютери.

- Трансформація зображень

Даний тип обробки найбільш широко поширений при аналізі зображень. Він заснований на впливі на функцію, що описує зображення операторами різного типу. До таких же перетворень зображення відноситься і фільтрація. В області такого поліпшення якості зображення досягнуті найбільші успіхи. Величезна кількість літератури присвячено саме цим видом обробки. При його використанні необхідні оцінки параметрів спотворень і це використовується при подальшій корекції зображення.

- Реконструкція або відновлення зображення

Зображення завжди формується деякими приладами, що включають в себе велику кількість елементів його перетворює. Таким чином, між предметом і зображенням існує система формування зображення, яка може дуже сильно спотворювати предмет. Впливом системи формування зображення обумовлені такі його характеристики як роздільна здатність, контраст в передачі різних просторових частот і т.п. В результаті ми завжди отримуємо зображення відмінне від функції описує об'єкт. При цьому знижується цінність отриманої інформації, отже, необхідно перетворити зображення так, щоб реконструювати (відновити) функцію, що описує об'єкт. Реконструкція зображення - це витяг деталей в спотвореному зображенні при відомій апріорної інформації про процес формування зображення і про об'єкт [7]. Як правило цей тип обробки відноситься до зворотних завдань математичної фізики, які мають на увазі отримання зображення, максимально

наближеного до об'єкта, з урахуванням властивостей системи, яка формує зображення.

В курсовій роботі був розроблений програмний продукт, в якому були реалізовані алгоритми відновлення зображень. Результатом виконання програми є відновлене зображення. Були реалізовані такі методи відновлення розмитих зображень, як фільтр Вінера та метод регуляризації Тихонова.

Актуальність роботи полягає в тому, що автоматичне збільшення якості зображень — це одна з найважливіших та найскладніших задач комп'ютерної графіки. Обробка зображень є важливою у роботі з відео та фото, у медицинських дослідженнях, у машинному навчанні та в задачах розпізнавання.

Курсова робота складається з трьох розділів, списку використаної літератури та додатку. У першому розділі описаний аналіз опрацьованої літератури та вибір напрямку досліджень. У другому розділі описуються теоретичні відомості про регуляційні алгоритми, що використовують апарат інтегральних рівнянь, та описується принцип роботи фільтра Вінера, з яким буде виконаний порівняльний аналіз. У третьому розділі описується реалізація алгоритмів в програмному продукті, розглядаються чисельні експерименти і аналіз результатів.

ПОСТАНОВКА ЗАДАЧІ

Нехай задача відновлення спотвореного зображення описується набором одновимірних інтегральних рівнянь Фредгольма I роду типу згортки:

$$Aw = \int_{-\infty}^{\infty} h(x-\xi)\, w_y(\xi)\, d\xi = g_y(x) + \delta g, a \le x \le b, c \le y \le d, \qquad (1)$$

або двовимірним інтегральним рівнянням Фредгольма I роду типу згортки:

$$Aw = \iint_{-\infty}^{\infty} h(x-\xi, y-\eta)\, w(\xi,\eta)\, d\xi\, d\eta = g(x,y) + \delta g, \qquad (2)$$

де h – ядро інтегрального рівняння, що являє собою функцію розсіювання точки, у більшості випадків просторово-інваріантної; A – лінійний інтегральний оператор; w і g – розподіл інтенсивності за істинним та спотвореним зображеннями відповідно; δg – похибка. В (1) вісь Ox спрямована вздовж змазування, а y відіграє роль параметра. Набір рівнянь (1) часто використовується в задачі змазування, а (2) – в задачі дефокусування зображення [2–5, 8].

Для вирішення задачі відновлення зображення необхідно розв’язати обернену задачу (1).

У роботі необхідно: ознайомитись з існуючими алгоритмами розв'язання задачі відновлення зображень, реалізувати їх на ЕОМ, дослідили якість та швидкість алгоритмів, зробити порівняльний аналіз та сформулювати висновки.

1 АЛГОРИТМИ ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ

1.1 Метод регуляризації Тихонова

Графічний процесор, як правило, записують у операторному вигляді. Оператор - це перетворення, яке здійснює трансформацію деякої функції в функцію.

Нехай існує двовимірний об'єкт (x,y), його двовимірне зображення $q(\xi,\eta)$ і A - оператор, який впливає на об'єкт.

Суть задачі полягатиме у знаходжені оберненого оператора A^{-1} при відомому операторі A, та використовуючи його отримати отримати реконструйований об’єкт:

$$A^{-1}[q(\xi,\eta)]=f*(x,y) \qquad (3)$$

де — $f*(x,y)$ зображення, яке максимально наближене до об'єкта.

Метод формування зображення проілюстровано на рисунку 1.1.

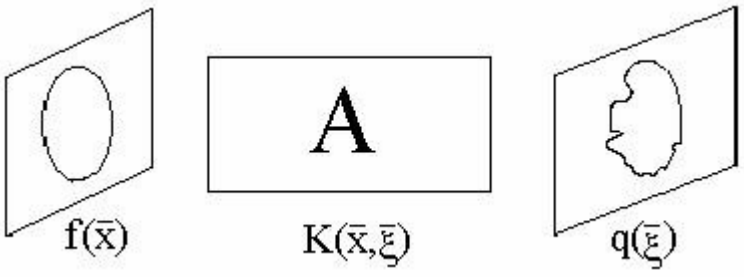

Рисунок 1.1 - Одноступінчатий метод формування зображення

Як правило, такий процес формування зображення описується оператором в інтегральній формі:

$$q(\bar{\xi})=\int K(\bar{x},\bar{\xi})\cdot f(\bar{x})\cdot d\bar{x}, \qquad (4)$$

де $K(\bar{x},\bar{\xi})$– ядро інтегрального оператора (визначається приладом, що формує зображення і способом формування зображення). Для оптичних систем знаходиться з теорії дифракції.

Як правило, при одноступенчатом методі формування при хорошій якості приладу зображення схоже на об'єкт. Така ситуація в мікроскопії, астрономії і т.д. Тому процес реконструкції, тобто редукції до ідеального приладу, в даному випадку не застосовується. Його використовують тільки в унікальних випадках, коли хочуть досягти надзвичайної чіткості.

Припустимо, що зображення деякого об'єкту спотворено прямолінійним рівномірним горизонтальним змазом (в результаті зсуву фотоапарата, самого об'єкта і т.д.). Зворотнє завдання відновлення розмитого зображення [1-5] зводиться до вирішення (в кожному у-му рядку зображення) некоректного одновимірного інтегрального рівняння Фредгольма I роду типу згортки:

$$\int_{-\infty}^{\infty} h(x-\xi)w_y(\xi)d\xi = g_y(x)+\delta g, \qquad (5)$$

$$\iint_{-\infty}^{\infty} h(x-\xi, y-\eta)w(\xi,\eta)d\xi\, d\eta = g(x,y)+\delta g. \qquad (6)$$

Для вирішення інтегральних рівнянь (5) і (6) найбільш підходить метод регуляризації Тихонова [2-7, 10]. Так як для інструментальної реалізації інтерес представляють чисельні методи, в даній роботі їм і приділено увагу. При чисельної реалізації методу регуляризації Тихонова для розв'язання оберненої задачі зазвичай використовуються метод перетворення Фур'є (ПФ) і метод квадратур [2, 3, 5-7]. Використовуються також методи ітерацій, наприклад, метод итеративної регуляризації Фрідмана [2, 7, 10].

Метод регуляризації Тихонова з використанням перетворення Фур'є застосуємо для рішення рівнянь типу згортки. Розглянемо окремий випадок IУ Фредгольма I роду - рівняння типу згортки одномірне (1) і двовимірне (2). Якщо IУ виду (1) при його чисельному рішенні методом квадратур вимагає

розміщення в комп'ютерній пам'яті матриці СЛАР, то для вирішення одновимірного рівняння типу згортки можливо застосовувати метод ПФ, який оперує лише з векторами, і це вимагає меншого обсягу пам'яті і часу вирішення. Особливо важливим є питання про пам'ять і часу при апаратній реалізації методу, в ще більшому ступені - для двовимірного рівняння (2).

Завдання змазування зводиться до вирішення одновимірного ІУ Фредгольма I роду типу згортки при кожному фіксованому значенні y, що грає роль параметра. У рівнянні (5) ФРТ виражається формулою:

$$h(x)=\begin{cases} 1/\Delta, -\Delta \le x \le 0 \\ 0, \textit{інакше} \end{cases}, \qquad (7)$$

де Δ — величина змазу.

Просторова инвариантность функції h має місце в разі, коли величина Δ не залежить від x. Якщо значення Δ залежить від x (приклад: змазування різні на знімку, зробленому нерухомим фотоапаратом, де представлені легкоатлети, які тікають з різною швидкістю), то рівняння задачі про змазування зображення буде мати вигляд:

$$\int_a^b h(x,\xi) w_y(\xi)\, d\xi = g_y(x) + \delta g, c \le x \le d. \qquad (8)$$

Як у випадку рівняння (5), так і (8), необхідно вирішити стільки самостійних (необразуючіх систему) одновимірних рівнянь, скільки рядків містить зображення.

Рішення рівняння (5) методом ПФ з регуляризації Тихонова має вигляд [2, 3, 5-7, 10]:

$$w_{\alpha y}(\xi) = \frac{1}{2\pi} \int_{-\infty}^{\infty} W_{\alpha y}(\omega) e^{-i\omega\xi} d\omega, \qquad (9)$$

де

$$w_{\alpha y}(\xi)=\frac{H(-\omega)G_y(\omega)}{|H(\omega)|^2+\alpha\omega^{2p}}, \tag{10}$$

$$H(\omega)=\int_{-\infty}^{\infty} h(x)e^{i\omega x}dx \quad , \tag{11}$$

$$G_y(\omega)=\int_{-\infty}^{\infty} g_y(x)e^{i\omega x}dx, \tag{12}$$

- перетворення Фур'є від $g_y(x)$ і $h(x)$ відповідно, $\alpha>0$ - параметр регуляризації, а $p\geq 0$ - порядок регуляризації (зазвичай 1 або 2). При комп'ютерної реалізації формул (5) і (6) усі безперервні перетворення Фур'є замінюються на дискретні, а також швидкі ПФ (БПФ) [1, 4, 6]. Число відліків по x і по ω вважаємо однаковим і рівним n.

Для вирішення ІУ (5) і (8) можна застосувати метод кінцевих сум (квадратур) з регуляризації Тихонова, згідно з яким інтеграл при кожному x замінюється кінцевою сумою з цілим кроком дискретизації (в пікселах) і виходить система лінійних алгебраїчних рівнянь (СЛАР) при деякому фіксованому y:

$$Aw_y=g_y, \tag{13}$$

де A — матриця розмірності nxn, що пов'язана з ФРТ (7), тобто $A_{ik}=h(x_i,\xi_k)$, де $x_i=\xi_i=1,2,\ldots,n$ - дискретні цілочисельні відліки за пікселами вздовж y-го рядка (n - число стовбців в зображенні). Зазначимо, що вектори w_y та g_y мають довжину n.

У методі регуляризації Тихонова замість (5) вирішується рівняння

$$(\alpha E+A^T A)w_{\alpha y}=A^T g_y, \tag{14}$$

де $\alpha > 0$ — параметр регуляризації, E — одиничний оператор (одинична матриця). Матриця $(\alpha E + A^T A)$ є квадратною, симетричною і позитивно-визначеною, СЛАР (14) має рішення і чисельно стійка [2, 3, 12].

Для того, щоб реалізувати метод регляризації Тихонова необхідно:

1. Визначити параметр регуляризації: за емпіричними даними або методами наведеними у [1, 7, 8].
2. Застосувавши метод квадратур, розрахувати матрицю А за формулою (13).
3. Знайти добуток одиничної матриці на параметр регуляризаії.
4. Знайти добуток транспонованої матриці А на матрицю А.
5. Знайти зворотню матрицю до суми величин, отриманих у кроках 3 та 4.
6. Результат, отриманий у кроці 5 домножити на транспоновану матрицю А та рядок спотвореного зображення.
7. Нормалізувати результат.

1.2 Фільтр регуляризації Фрідмана

Нехай ядро h рівняння (1) симетрично, тобто $h(x,y)=h(|x|,|y|)$.

Тоді метод Фрідмана для вирішення ІУ (1) описується наступним співвідношенням [10, 11, 13]:

$$w_k(x,y)=w_{k-1}(x,y)+v\left[g(x,y)-\iint_{ac}^{bd} h(x-\xi,y-\eta)\,w_{k-1}(\xi,\eta)\,d\xi\,d\eta\right], \qquad (15)$$

де $w_{k-1}(x,y)$ та $w_k(x,y)$ - рішення, отримані в $(k-1)$ та k ітераціях (наближення) відповідно, при цьому $w_0(x,y)$ - початкове наближення; $k=1,2,3\ldots$ - номер ітерації; v - параметр, що задовольняє умові $0\le v\le 2/\|A\|$, де $\|A\|=\left\{\iint_{ac}^{bd} h^2(x,y)\,dx\,dy\right\}^{0.5}$ - норма ФРТ.

Для задач дефокусування $w_k(x,y)$ – зображення, що отримане в k ітерації, а $g(x,y)$ – права частина рівняння (спотворене зображення). Метод Фрідмана вельми чутливий до початкового наближення, яке при реалізації зазвичай вважається рівним $w_0(x,y)=0$. Поставивши собі за якимось (не обов'язково нульовим) початковим наближенням і виконуючи ітераційний процес (3), можна отримати наближення до шуканого рішення (істинного зображення). Але, оскільки задача некоректна, а права частина зазвичай зашумлена, то до деякого номера ітерацій $\hat{k}$ процес зазвичай сходиться до вирішення, після чого він починає розходитися [10]. Тому потрібно ввести критерій зупинки процесу, тобто вибору числа ітерацій $\hat{k}$, яке відіграє роль параметра регуляризації.

Основна ідея цього методу полягає у побудові ітераційної схеми, що сходиться до точного розв’язку рівняння за відсутності помилок δ та θ правої частини g та оператора A відповідно, і процесу ітерацій, що в перериванні розходиться при $\eta\equiv(\delta,\theta)\neq 0$ при деякому числі ітерацій $m=m(\eta)$ (яке виступає у ролі параметра регуляризації) такому, що $\|w^{m(\eta)}-w\|\to 0$ при $\eta\to 0$.

Метод ітерацій Фрідмана є одним із найпростіших і достатньо ефективних. Він належить до групи стаціонарних ітераційних методів розв’язання лінійного операторного рівняння першого роду

$$Aw=g,\ w\in H_1,\ g\in H_2\ , \qquad (16)$$

де $A:H_1\to H_2$ - лінійний цілком неперервний оператор, H_1 та H_2 – гільбертові простори.

1.3 Фільтр Вінера

Сформулюємо постановку задачі у термінах згортки.

Введемо множину $Z_m^n = \{ i \in Z : m \le i \le n \}$.

Нехай спотворене зображення позначено як $g : Z_0^w \times Z_0^h \to Z_0^{255}$, де w — ширина зображення, h — висота зображення, 255 — кількість можливих кольорів пікселів.

Оригінальне зображення позначено як $f : Z_0^w \times Z_0^h \to Z_0^{255}$.

Функція спотворення визначена $h : Z_{-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \times Z_{-\lfloor l/2 \rfloor}^{\lfloor l/2 \rfloor} \to R$, де k - ширина матриці ядра спотворення, l - висота матриці ядра спотворення. Мають виконуватися наступні умови: $k \le w$, $l \le h$, $k = 2 * N + 1$, $l = 2 * N + 1$, $k * l \ge 3$.

Функція адитивного шуму визначена як $n : Z_0^w \times Z_0^h \to R$.

Функція нормування позначена $N : R \to Z_0^{255}$.

x, y - координати ширини та висоти пікселя у зображенні.

Тоді модель процесу спотворення зображення можна описати таким чином:

$$g(x, y) = N\big((h * f)(x, y) + n(x, y)\big). \qquad (17)$$

Розглянемо модель спотворення зображення (17).

Для того, щоб розв'язати задачу відновлення, необхідно з формули (17) знайти $f(x, y)$. Вирішення цієї задачі може призвести до розв'язування великої системи рівнянь, що не є зручним способом знаходження $f(x, y)$ та значно сповільнює виконання програми.

Для подолання цієї проблеми використовують перетворення Фур'є. Перетворення Фур'є — це операція, що ставить у відповідність однієї функції дійсних чисел другу функцію дійсних чисел. Нова функція описує коефіцієнти при розкладі існуючої функції на елементарні складові, тобто гармонічні коливання з різними частотами.

Перетворення Фур'є деякої функції f дійсної змінної є інтегральним та задається формулою:

$$\hat{f}(\omega)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)e^{-ix\omega}dx. \qquad (18)$$

Для перетворення Фур'є функцій заданих на просторі $\mathbb{R}^n$ використовується формула багатовимірного інтегрального перетворення:

$$\hat{f}(\omega)=\frac{1}{(2\pi)^{n/2}}\int_{R^n} f(x)e^{-ix\omega}dx. \qquad (19)$$

Зворотне перетворення у такому випадку задається формулою:

$$f(x)=\frac{1}{(2\pi)^{n/2}}\int_{R^n} \hat{f}(\omega)e^{ix\omega}d\omega, \qquad (20)$$

У деяких джерелах [5, 11] відрізняється перед інтегральний коефіцієнт для багатовимірного перетворення, що може незначно вплинути на вихідний результат.

Зазвичай для обробки сигналів (наприклад, у стисканні звука в MP3 або у стисненні зображень в JPEG) а також в інших областях, пов'язаних з аналізом частот до дискретному сигналі використовують дискретне перетворення Фур'є (DFT - Discrete Fourier Transform). Дискретне перетворення приймає на вхід дискретну функцію або вибірку значень з неперервної функції. Дискретне перетворення використовують для виконання операції згортки, яка наявна у задачі відновлення.

Пряме дискретне перетворення задається формулою:

$$X_k=\sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i}{N}kn} \quad k=0,\dots N-1. \qquad (21)$$

Аналогічна формула для зворотного перетворення:

$$x_n=\frac{1}{N}\sum_{n=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \quad n=0,\dots N-1. \qquad (22)$$

Дискретне перетворення Фур'є є лінійним перетворенням, яке переводить вектор тимчасових відліків в вектор спектральних відліків тієї ж довжини.

Для того, щоб обчислити згортку двох функцій дійсних змінних f та g, використаємо формули для прямого та зворотного дискретного перетворення Фур'є:

$$f x g = BFT\left(FFT(f) * FFT(g)\right), \tag{23}$$

де *FFT* – операція прямого перетворення Фур'є (Forward Fourier Transform), *BFT* – операція зворотного перетворення Фур'є (Backward Fourier Transform)

Використавши формулу прямого перетворення Фур'є, перепишемо задачу (17) таким чином:

$$G(u,v) = H(u,v)F(u,v) + N(u,v), \tag{24}$$

де $G(u,v), H(u,v), F(u,v), N(u,v)$ – функції g, h, f, n після прямого перетворення Фур'є відповідно, u, v — координати ширини та висоти відповідно.

Найпростіший спосіб приблизно знайти $F(u,v)$ – розділити рівняння (24) на $H(u,v)$.

Отримаємо таку оцінку $\hat{F}(u,v)$ оригінального зображення:

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}. \tag{25}$$

Проте такий спосіб майже не використовується на практиці та дає

погані результати. Якщо функція $H(u,v)$ приймає близькі до нуля значення (що майже завжди так), то вклад останнього доданку буде домінуючим. Щоб уникнути цього ефекту, значення функції $N(u,v)$ теж мають бути максимально близькими до нуля. Тому додання невеликого шуму до зображення призводить до значних візуальних недоліків.

Тому розробляються підходи, які враховують наявність шуму на зображенні. Один із самих відомих та ефективних – фільтр Вінера. Фільтр Вінера скорочує рівень випадкового шуму на зображеннях.

Ідея методу заснована на тому, що адитивний шум – це стаціонарний випадковий процес, що не залежить від розташування пікселя. Алгоритм мінімізує квадратичну похибку між початковим та відновленим зображенням.

Мета роботи фільтру полягає у пошуку такої оцінки $\hat{F}(u,v)$ для оригінального зображення $F(u,v)$, щоб середньоквадратичне відхилення цих зображень було мінімальне.

При побудові фільтру Вінера ставиться задача мінімізації середньоквадратичного відхилення відновленого зображення та початкового:

$$E\left\{\left[F(u,v)-\hat{F}(u,v)\right]^2\right\}=min, \qquad (26)$$

де E – математичне сподівання.

Шляхом перетворень можна показати, що мінімум досягається коли спотворююча функція визначається наступним чином:

$$H(u,v)=\frac{S_{f'}(u,v)}{S_f(u,v)}, \qquad (27)$$

де S_f та $S_{f'}$ — енергетичні спектри оригінального та відновленого зображення.

Спектральна щільність сигналу визначається відношенням:

$$S_f(v)=FFT\left[R(\omega)\right], \tag{28}$$

де $R(\omega)=\int_{-\infty}^{\infty} F(x)F(x-\omega)\,\mathrm{dx}$ – автокореляційна функція

Взаємна спектральна щільність сигналу:

$$S_{f'}(v)=FFT\left[R(\omega)\right], \tag{29}$$

де $R(\omega)=\int_{-\infty}^{\infty} F(x)\hat{F}(x-\omega)\,\mathrm{dx}$ – функція взаємної кореляції

Відновлення зображення відбувається з використанням наступного відновлюючого перетворювача:

$$R(u,v)=\frac{1}{H(u,v)}\,\frac{\left|H(u,v)^2\right|}{\left|H(u,v)^2\right|+S_n(u,v)/S_f(u,v)}. \tag{30}$$

Знайдемо наближене відновлене зображення $\hat{F}(u,v)$ за формулою:

$$\hat{F}(u,v)=R(u,v)G(u,v). \tag{31}$$

Параметри $S_n(v), S_f(v)$ являють собою енергетичні спектри шуму та зображення відповідно. Знаходження цих параметрів є окремою підзадачею. Існує декілька способів їх визначення.

Відношення $S_n(u,v)/S_f(u,v)$ можна приблизно визначити як відношення сигналу до шуму (Signal-to-Noise Ratio, SNR).

$$SNR = \frac{P_{signal}}{P_{noise}} = \left(\frac{A_{signal}}{A_{noise}} \right)^2, \quad (32)$$

де P_{signal} – середня потужність, P_{noise} – середньоквадратичне значення амплітуди. Обидва сигнали вимірюються в смузі пропускання системи. Зазвичай відношення сигнал / шум виражається в децибелах (дБ):

$$SNR = 10\,lg\left(\frac{P_{signal}}{P_{noise}} \right) = 20\,lg\left(\frac{A_{signal}}{A_{noise}} \right). \quad (33)$$

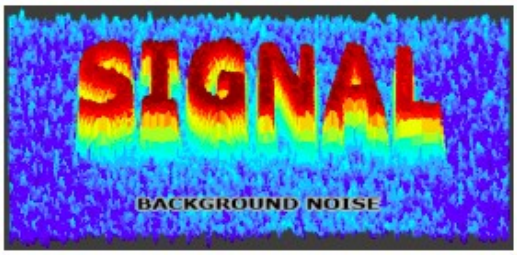

Рисунок 1.2 - Сигнал значно більший за шум

У наведеному вище зображенні (рис. 1.2) співвідношення сигнал-шум досить велике, щоб чітко відокремити інформацію в зображенні від фонового шуму. Низьке відношення сигналу до шуму могло б привести до появи зображення, де сигнал і шум можна порівняти, і тому їх складніше відрізнити один від одного (рис. 1.3).

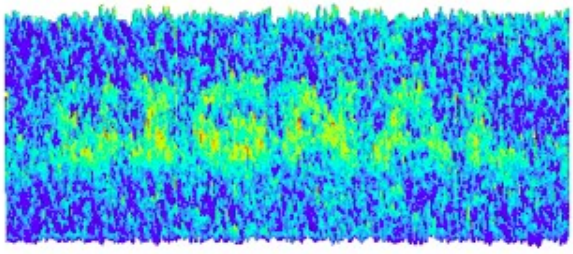

Рисунок 1.3 - Сигнал подібний до шуму

Проводячи аналіз зображення, отримуємо дані про середнє значення яскравості і стандартному відхиленні. На основі цих даних проводиться розрахунок відношення сигналу до шуму за формулою:

$$SNR=\frac{M}{\sigma}, \tag{34}$$

де M − середнє значення яскравості; σ − стандартне відхилення.

Стандартне відхилення є мірою шуму (тобто наскільки вибрані пікселі можуть відрізнятися від вищезазначеної середньої величини яскравості).

Існує альтернативне визначення коефіцієнтів $S_n(u,v)$ та $S_f(u,v)$:

$$S_n(u,v)=M\left[\left\|N(u,v)\right\|^2\right], \tag{35}$$

$$S_f(u,v)=\left\|F(u,v)\right\|, \tag{36}$$

де M — математичне сподівання, $F(u,v)$ — оригінальне зображення, $N(u,v)$ — значення шуму.

Шум, що спотворює зображення, вважаємо адитивним білим і не корельованим із зображенням. Відомо, що основна інформація зображення зосереджена в області нижніх частот. В області високих частот значення спектра зображення близькі до нуля і внеском першого доданка можна знехтувати. Таким чином, в деякій високочастотної області спектр зображення близький до спектру дискретного білого шуму $S_n(u,v)\approx\sigma_n^2$.

Для оцінки середньоквадратичного відхилення σ_η використовуємо формулу:

$$\sigma_n\approx\sqrt{\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}\left|G(u,v)^2\right|}, \tag{37}$$

де Ω – високочастотна область, що розглядається в даний момент, $G(u,v)$ – спектр спотвореного зображення.

Щоб застосувати фільтр Вінера до зображення, потрібно:

1. Перевести матрицю цілих чисел зображення $g(x,y)$ у матрицю комплексних чисел.

2. Визначити спотворюючу функцію $h(x,y)$ у вигляді дискретного набору значень.

3. Застосувати пряме перетворення Фур'є до матриці зображення $g(x,y)$ та матриці спотворюючої функції $h(x,y)$ за формулою (24), отримавши матриці $G(u,v)$ та $H(u,v)$.

4. Розрахувати відношення $S_n(u,v)/S_f(u,v)$ як відношення сигналу до шуму (SNR) за формулою (34), або окремо розрахувати $S_n(u,v)$ та $S_f(u,v)$ за формулами (35, 36).

5. Розрахувати наближене відновлене зображення $\hat{F}(u,v)$ за формулою (2.5).

6. Застосувати зворотне перетворення Фур'є за формулою (22) до зображення $\hat{F}(u,v)$, отримавши вихідне зображення $f(x,y)$.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ

2.1 Опис програми

Постановка задачі

Нехай на вхід програми задане цифрове зображення розміром $N \times M$. Величини N та M – розміри піксельної сітки, на елементах якої задані значення g_{ij}, що характеризують яскравість або колір пікселя (i, j) у зображенні g, що було зіпсоване функцією h. Задачею відновлення зображення є отримання відновленого зображення f з відомого спотвореного зображення g та даних про функцію спотворення h з урахуванням функції шуму n.

Опис інтерфейсу

Користувачу надається зручний та простий у використанні інтерфейс.

Спочатку користувач має натиснути кнопку “Open image”, перейти до потрібної директорії та обрати тип зображення (програма працює з файлами типу bitmap та jpeg), потім обрати оригінальне зображення для завантаження. Зображення буде виведено у верхню ліву частину. Далі користувачу пропонується змоделювати зіпсоване зображення, обравши розмір ядра h у вікні “Kernel size” та його тип, натиснувши одну з кнопок “Gaussian filter”, “Sharpen Filter”, “Motion Filter”. Користувач має можливість додати розмиття з радіусом певного розміру до вхідного зображення, підвищення чіткості з певною силою або зсув у довільному напрямку на довільну відстань. Зіпсоване зображення буде виведено у правій верхній частині.

Для відновлення зіпсованого зображення користувачу пропонується два методи: фільтрація Вінера з відомим ядром та метод регуляризації Тихонова. У першому випадку інформація введена користувачем про спосіб зіпсування оригінального зображення буде використовуватися у роботі

метода відновлення, у другому випадку ця інформація використається лише для отримання зіпсованого зображення.

Після роботи метода у лівому нижньому вікні буде виведено зображення ядра, яким було зіпсовано вхідне зображення. У правому нижньому вікні буде зображене вихідне зображення.

Для взаємодії з користувачем був розроблений клас Form, у якому оброблюються такі дії користувача як натиск на кнопку та відкриття файлового діалогу, та відправляється команда на виконання потрібного функціоналу в залежності від дії користувача.

Для роботи із зображеннями був розроблений клас ImageHelper, у якому містяться допоміжні методи обробки зображень, такі як: FFT2 та FFT — пряма Фур'є-трансформація для двовимірного набору даних та одновимірного набору даних відповідно; BFT2 та BFT — зворотна Фур'є-трансформація для двовимірного набору даних та одновимірного набору даних відповідно; ConvertTo8bpp — перетворення вхідного зображення до 8-бітного формату; Crop — вирізання певної ділянки зображення; GetAverage — розрахунок середнього значення сигналу; GetComplexImageFromMatrix — перетворення набору комплексних або дійсних даних у зображення; GetCoreImage — отримання ядра спотворення у вигляді зображення; GetDispersion — розрахунок дисперсії; GetExtended — отримання розширеного до деякого розміру зображення з вхідного; GetMSE, GetPSNR, GetSNR — розрахунки критеріїв якості MSE, PSNR та SNR відповідно; Rotate — повернути зображення; ToGray — конвертація кольорів зображення у чорно-білий спектр; ToVector — конвертація набору даних з матричного вигляду до векторного; BitmapToMatrix — конвертація зображення у двовимірний масив цілих чисел.

Метод регуляризації Тихонова

Для роботи алгоритму методу регуляризації Тихонова був розроблений клас TikhonovFilter, який містить основний функціонал. До

головного методу Filter передається спотворене зображення та тип ефекту, який було застосовано до оригінального зображення (розмиття, змаз, тощо).

Застосуємо реалізацію методу регуляризації Тихонова використовуючи метод квадратур:

$$w_\alpha=(\alpha E+A^T A)^{-1}A^T g \quad , \qquad (38)$$

де $\alpha>0$ - параметр регуляризації, E - одинична мтариця, A^T - транспонована матриця.

Для цього розрахуємо матрицю А використовуючи метод GetA. Для кожного елемента матриці А застосуємо визначену функцію спотворення зображення h. У методі GetH реалізуємо формулу (7), та повернемо значення $1/\Delta$ або 0 в залежності від значень x та ξ . Розрахуємо одиничну матрицю Е у методі GenerateEMatrix. Розрахуємо транспоновану матрицю А використовуючи бібліотечний метод Matrix.Transpose(A). Наступним кроком буде визначення добутку одиничної матриці на коефіцієнт регуляризації використовуючи метод Multiply(). Коефіцієнт регуляризації статично зафіксований у класі TikhonovFilter. Далі розраховуємо добуток транспонованої матриці А на матрицю А за допомогою методу Multiply(). Додамо отримані результати за допомогою методу Add(). Отриману матрицю інвертуємо за допомогою методу Inverse(). Домножимо отриману зворотню матрицю на транспоновану методом Multiply(). Конвертуємо вхідне зображення використовуючи метод BitmapToMatrix класу ImageHelper, що був створений для базової роботи із зображеннями. Перемножимо отримані результати із векторним уявленням вхідного зображення. Далі отримані результати переведемо у проміжок допустимих значень, тобто виконаємо нормалізацію. Для нормалізації буде застований метод Normalize, що переводить вхідний масив у масив такої самої розмірності, але із цілочисельними значеннями у проміжку [0, 255]. Повернемо фінальний

результат у вигляді зображення типу Bitmap використовуючи метод ToBitmap(). Відобразимо результат користувачу.

Побудова трансформації Фур'є

Окремою підзадачею є побудова та застосування дискретних трансформацій Фур'є, що ускладнює реалізацію алгоритмів та збільшує час обробки зображень. Для роботи фільтрації Вінера також існує необхідність у використання прямого та зворотного дискретних перетворень Фур'є.

Нехай дані зображення вже перетворено до комплексного формату та надано у векторному вигляді. У допоміжному класі ImageHelper існує метод FFT2, що виконує двовимірне пряме дискретне перетворення Фур'є. Він перетворює вхідний вектор одновимірним дискретним перетворенням, та повертає результат у матричному вигляді. Одновимірне дискретне перетворення у методі FFT відбувається рекурсивно. На початку оцінюється розмір вхідного вектору. Якщо його розмір дорівнює двом, то вихідний вектор розраховується як сума та різниця елементів вхідного вектора, та робота метода закінчується. Якщо розмір вектора більший, то його елементи розділяються на два вектора — вектор елементів з парним та непарним індексом. Кожний з векторів рекурсивно оброблюється цим же методом. З отриманих після цієї операції векторів формується новий вектор за формулою:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i}{N} kn} \quad k = 0, \dots N-1. \qquad (39)$$

Після завершення дій алгоритму з використанням векторів оброблених прямою дискретною трансформацією Фур'є необхідно трансформувати отриманий результат зворотною дискретною трансформацією Фур'є. Зворотна дискретна трансформація відбувається з використанням методу BFT2 класу ImageHelper. Він перетворює вхідний

вектор одновимірним дискретним перетворенням, та повертає результат у матричному вигляді. Одновимірне дискретне перетворення у методі BFT за формулою:

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \quad n = 0, \dots N-1. \tag{40}$$

Розрахунок відбувається спряженням вхідного вектора та передачею його на рекурсивне перетворення до метода FFT. Після завершення дії метода FFT розраховується коефіцієнт, що дорівнює частці від ділення одиниці на довжину вектора. Вихідний масив заповнюється добутками спряжених значень векторів на розрахований коефіцієнт. Результат повертається перетвореним до матриці.

Фільтр Вінера

Для роботи алгоритму фільтрації Вінера був розроблений клас WienerFilter, який містить основний функціонал. До головного методу Filter передається спотворене зображення та тип ефекту, який було застосовано до оригінального зображення (розмиття, змаз, тощо). Дані з отриманого зображення перетворюються до комплексного вигляду шляхом нормалізації значень до проміжку [0, 1] та записом їх у дійсну частину комплексного числа.

Для отримання матриці ядра необхідно застосувати метод GetCore. Залежно від типу ядра, для знаходження його матриці викликається метод GetCore в одному із класів фільтрів GaussianFilter, SharpenFilter або MotionFilter. Із властивості, що визначає розмір ядра, відповідна інформація переходить до метода GetCore класу WienerFilter. Формула Вінера передбачає виконання множення значень, отриманих з елементів матриці ядра, на елементи матриці зображення, проте така операція неможлива, оскільки матриці зазвичай є різного розміру. Для подолання цієї проблеми необхідно

збільшити матрицю ядра до розмірів зображення, та заповнити нулями нові комірки матриці, залишивши фактичні значення ядра у центральній частині. Метод GetCore повертає отриману збільшену матрицю ядра у вигляді матриці дійсних чисел.

Для подальшої роботи метода необхідно перетворити матрицю ядра до комплексного зображення. Для цього викликається метод допоміжного класу ImageHelper GetComplexImageFromMatrix, що створює комплексне зображення на основі матриці дійсних чисел, записуючи ці значення до дійсної частини комплексних чисел.

У методі GetSNR розраховується середнє значення яскравості пікселів та стандартне відхилення методами GetAverage та GetDispersion. Метод повертає частку від ділення цих величин.

Отримане зображення та ядро необхідно трансформувати прямим дискретним трансформуванням Фур'є. Для цього необхідно перевести їх дані до векторного вигляду у методі ToVector, де послідовно конкатенуються всі дані з матриці, та передати отриманий вектор до методу FFT2. Отримані дані знову необхідно перевести до типу комплексного зображення методом GetComplexImageFromMatrix.

Після того, як необхідні дані було перетворено до Фур'є-образів, виконується основна формула фільтру Вінера (24) у методі GetF. Дані вихідного зображення отримуються розрахуванням цієї формули для кожного пікселя, та переведенням їх у формат комплексного зображення.

Після розрахунку оцінки оригінального зображення її необхідно перевести зворотною трансформацією Фур'є. Для цього, як і раніше, переведемо дані з матричного вигляду до векторного методом ToVector, розрахуємо методом BFT2 зворотну трансформацію Фур'є та перетворимо отримані дані до формату комплексного зображення. Комплексне зображення необхідно перевести до дійсного зі значеннями у проміжку [0, 255] шляхом виклику методу ToBitmap, який проводить денормалізацію результатів. Отримане зображення віддається на вихід для використання користувачем.

2.2 Опис розрахункових експериментів

Метод регуляризації Тихонова

Застосуємо метод регуляризації Тихонова із розмиттям та зсувом.

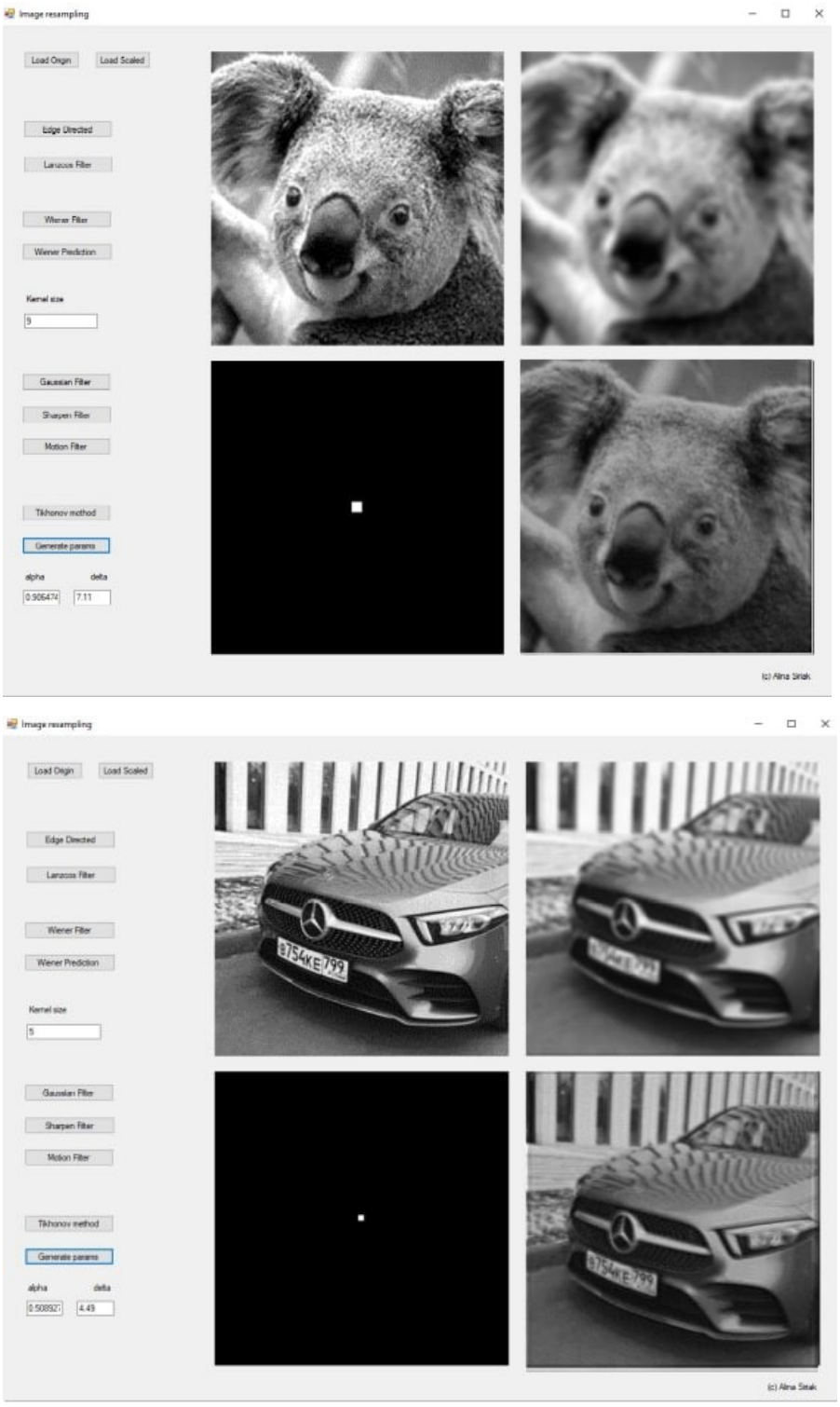

Рис. 2.1 - Розмиття з Гаусом та застосування методу регуляризації Тихонова

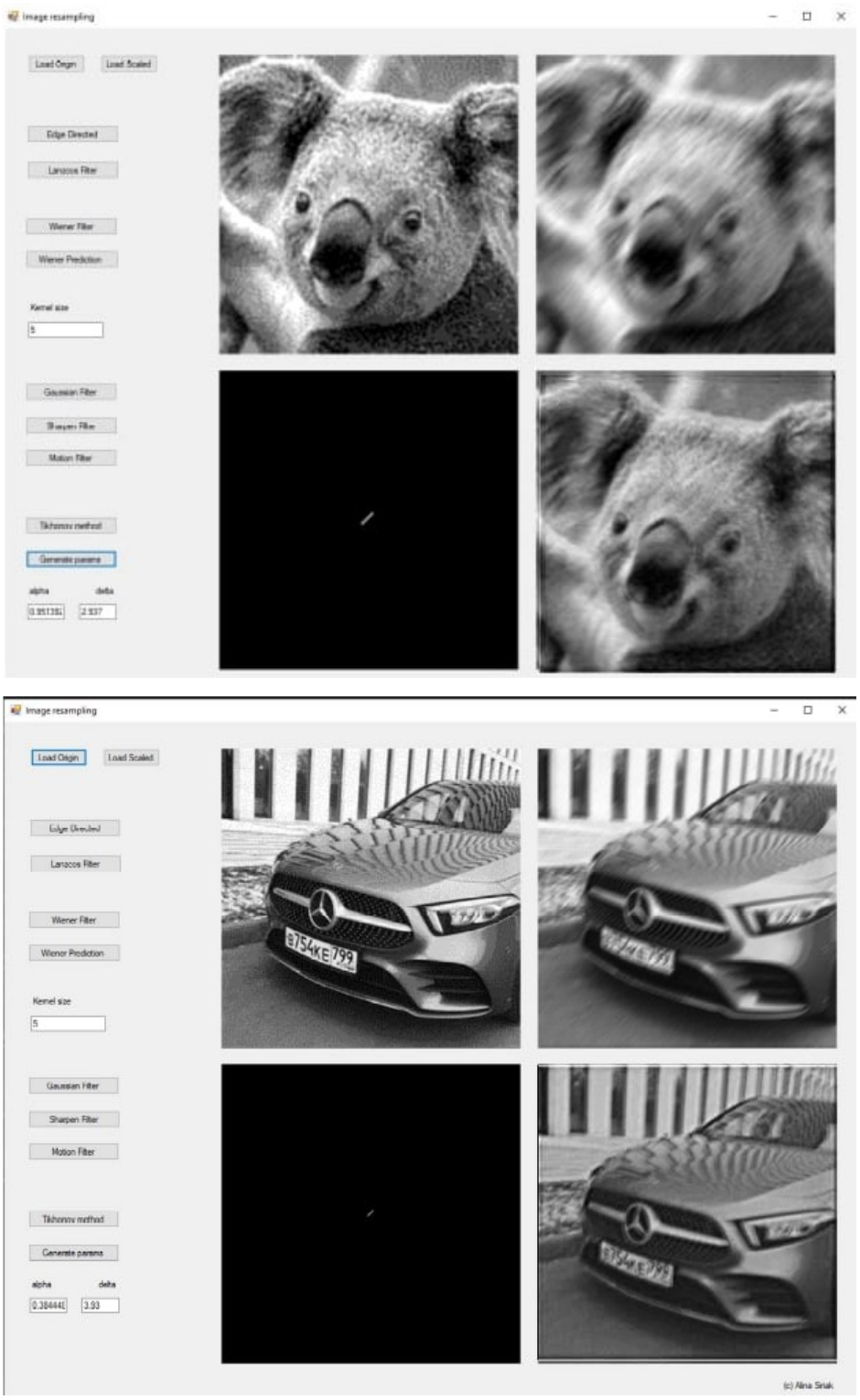

Рис. 2.2 - Змаз по діагоналі та застосування методу регуляризації Тихонова

Фільтр Вінера

Застосуємо фільтр Вінера до зображень із розмиттям та зсувом.

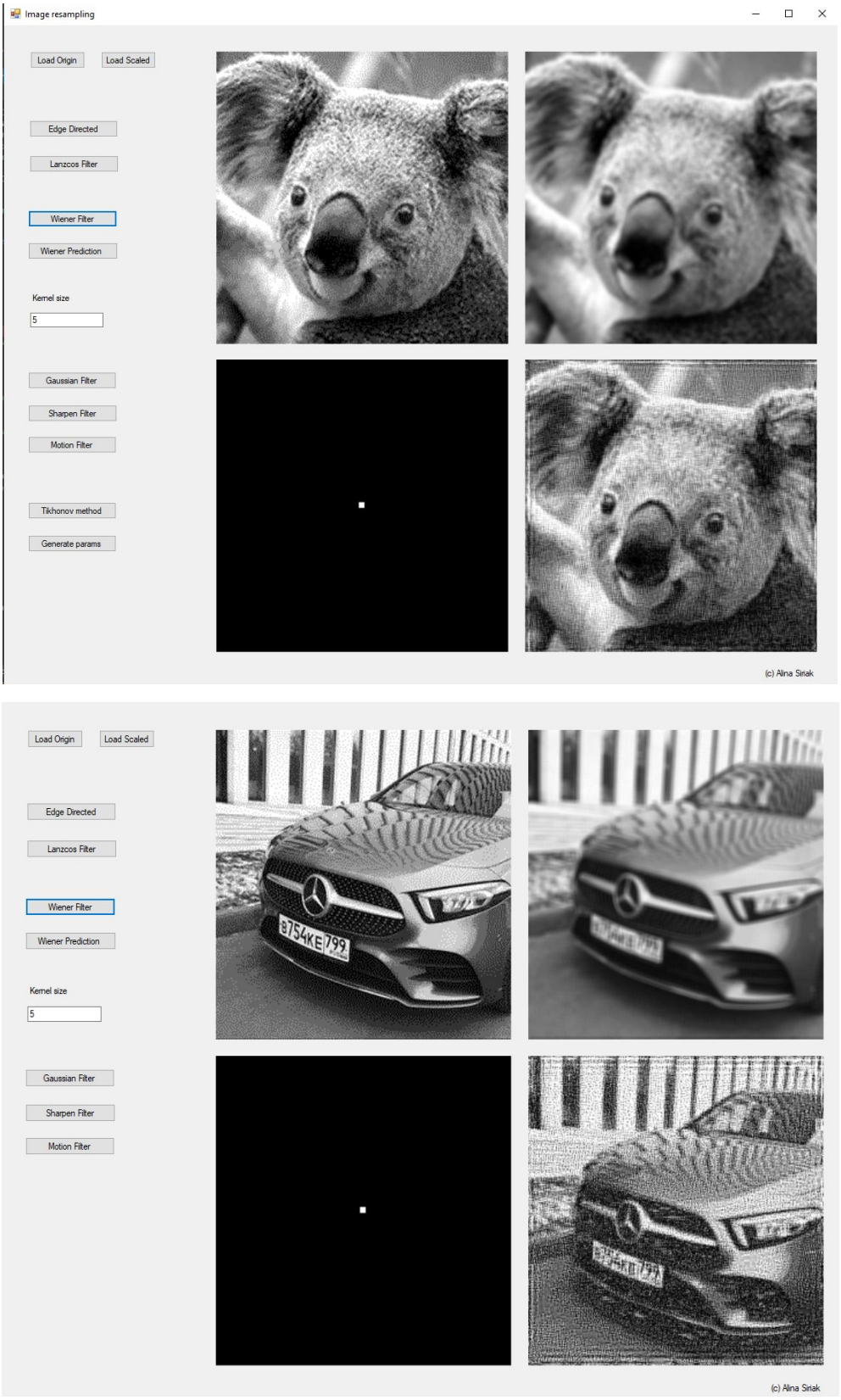

Рисунок 2.3 - Розмиття з Гаусом та застосування фільтру Вінера

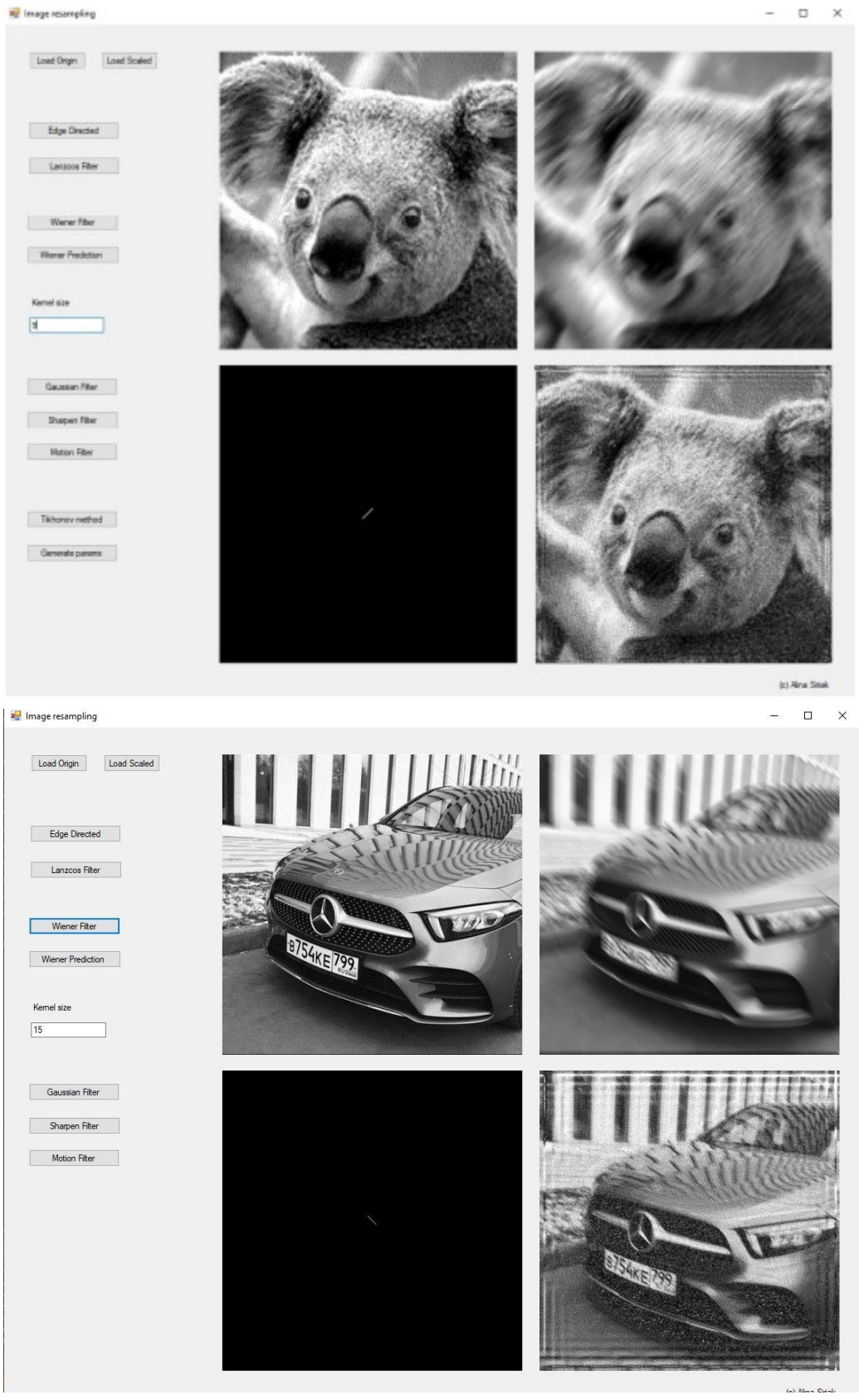

Рисунок 2.4 - Змаз по діагоналі та застосування фільтру Вінера

2.3 Порівняльна характеристика алгоритмів

Порівняльна характеристика була проведена за такими ознаками, як швидкість роботи алгоритмів та якість вихідного зображення.

Порівняємо метод регуляризації Тихонова та фільтрацію Вінера.

Швидкість роботи оцінена для зображень розмірів nxn.

Таблиця 2.1 Швидкість роботи алгоритмів для вхідних даних різної розмірності

Розмиття:

	Метод регуляризації Тихонова	Фільтр Вінера
$n = 128$	2.3823	1.8783
$n = 256$	7.1925	4.5211
$n = 512$	13.5296	9.3873

Змаз:

	Метод регуляризації Тихонова	Фільтр Вінера
$n = 128$	2.8489	1.5483
$n = 256$	6.9934	4.9151
$n = 512$	12.2109	9.1267

Для порівняння якості зображення розрахуємо PSNR отриманих зображень – відношення між максимально можливою потужністю сигналу і потужністю спотворюючого шуму.

Чим більше значення PSNR для зображення, тим якіснішим воно вважається.

Таблиця 2.2 Оцінка якості отриманих результатів для різних типів спотворень

Розмиття:

	Метод регуляризації Тихонова	Фільтр Вінера
$n = 128$	19.75	19.91
$n = 256$	18.44	18.36
$n = 512$	18.01	18.79

Зсув:

	Метод регуляризації Тихонова	Фільтр Вінера
$n = 128$	21.32	21.83
$n = 256$	19.91	20.99
$n = 512$	19.48	19.36

Візуальне порівняння.

Результат роботи алгоритмів наведений на рис.2.5 та рис.2.6.

а) б) в)

Рисунок 2.5. - а) розмите зображення б) відновлене методом регуляризації Тихонова зображення в) відновлене фільтром Вінера зображення

а) б) в)

Рисунок 2.6. - а) змазане зображення б) відновлене методом регуляризації Тихонова зображення в) відновлене фільтром Вінера зображення

На підставі отриманих результатів можна побачити, що метод квадратур з регуляризацією Тихонова та фільтрація Вінера дозволяють найбільш точно і якісно відновлювати зображення. Вихідне зображення у обох випадках мають майже однакову якість, для дефектів типу змазу працюють трохи краще. При значних дефектах зображення після відновлення помітний незначний ефект дзвону та зміна контрасності, проте такі ефекти можуть бути легко виправлені.

Що стосується затраченого часу на виконання алгоритмів, метод регуляризації Тихонова виявився більш ресурсоємким. Реалізація алгоритма із застосуванням методу квадратур вимагає виконання операцій множення матриць, що призводить до збільшення затраченого часу. Особливо цей недолік помітний на зображеннях великих розмірів.

Фільтр Вінера працює досить довго, при збільшенні розмірів вхідних даних швидкість помітно зменшується. Фільтр є складним у реалізації, потребує багато розрахунків та інформації про спотворення. Проте фільтр відновлює зображення досить якісно, незалежно від типу спотворення та його потужності. Навіть при дуже значних спотвореннях вдається відновити дрібні деталі, якість отриманого зображення є високою.

ВИСНОВКИ

При написанні даної курсової роботи були розглянуті і проаналізовані різні методи відновлення зображень та детально описані алгоритми їх роботи. Була вирішена математична задача знаходження рішення інтеграла Фредгольма I роду. Був створений програмний продукт, який показав роботу таких алгоритмів, як параметричний фільтр Вінера та метод регуляризації Тихонова із використанням методу квадратур, наведений опис цього продукту, а також представлена порівняльна характеристика результатів.

З отриманих результатів можна зробити наступні висновки:

- Обидва алгоритми точно та якісно відновлюють спотворені зображення.
- Метод Тихонова дає кращі результати для дефектів типу змазу, а також є простішим у реалізації. Проте він працює довше через велику кількість операцій множення матриць. Робота над оптимізацією методу регуляризації Тихонова являє собою інтерес для восконалення у наступних роботах.
- Фільтрація Вінера є складним у реалізації та потребує багато розрахунків. Проте фільтр відновлює зображення досить якісно, незалежно від типу спотворення та його потужності. Навіть при дуже значних спотвореннях вдається відновити дрібні деталі, якість отриманого зображення є високою.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Арефьева М.В., Сысоев А.Ф. Быстрые регуляризирующие алгоритмы цифрового восстановления изображений // Вычислит. методы и программирование. - 1983. - Вып. 39. - С. 40—55.
2. Воскобойников Ю.Е., Литасов В.А. Устойчивый алгоритм восстановления изображения при неточно заданной аппаратной функции // Автометрия. - 2006. - Т. 42. № 6. - С. 3–15.
3. Гарт Л.Л. Оптимізація обчислень в регуляризаційних алгоритмах розв’язання задачі відновлення розмитих зображень // Первый независимый научный вестник — 2015. - #1 — С. 165 -170
4. Гонсалес Р., Вудс Р. Цифровая обработка изображений. // М.: Техносфера - 2006. – С. 1072
5. Дайнеко М.В., Сизиков В.С. Восстановление смазанных под углом и зашемленных изображений без учета граничных условий // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики — 2010. - № 4(68)
6. Кирьянов К.А., Сизиков В.С. Применение сигнальных микропроцессоров в задачах реконструкции искаженных изображений. // Изв. вузов. Приборостроение. – 2011. – Т. 54. – № 7. – С. 20–26.
7. Сизиков В.С., Белов И.А. Реконструкция смазанных и дефокусированных изображений методом регуляризации // Оптический журнал. – 2000. – Т. 67. – № 4. – С. 60–63.
8. Сизиков В.С., Римских М.В., Мирджамолов Р.К. Реконструкция смазанных и зашумленных изображений без использования граничных условий // Оптический журнал. – 2009. – Т. 76. – № 5. – С. 38–46.

9. Сизиков В.С. Прием «усечение–размытие–поворот» для восстановления искаженных изображений // Оптический журнал. – 2011. – Т. 78. – № 5. – С. 18–26.
10. Сизиков В.С., Экземпляров Р.А. Последовательность операций при фильтрации шумов на искаженных изображениях // Оптический журнал. - 2013. - Т. 80. № 1. - С. 39–48.
11. Сизиков В.С. Обратные прикладные задачи и MatLab. // СПб: Лань - 2011. - С. 256
12. Тихонов А.Н., Гончарский А.В., Степанов В.В. Обратные задачи обработки фотоизображений // Некорректные задачи естествознания – М.: Изд-во МГУ - 1987. – С. 185–195.

ДОДАТОК

Лістинг програми

```
using System;
using System.Drawing;
using System.Linq;
namespace ImageInterpolation.Filtering
{
   public static class GaussianFilter
   {
      public static int BlurSize { get; set; }
      public static Bitmap Blur(Bitmap initialImage, int blurSize)
      {
         BlurSize = blurSize;
      var extendedImage = ImageHelper.GetExtended(initialImage, BlurSize);
         var f = new double[3][,];
         f[0] = new double[extendedImage.Width, extendedImage.Height];
         f[1] = new double[extendedImage.Width, extendedImage.Height];
         f[2] = new double[extendedImage.Width, extendedImage.Height];
         for (int i = 0; i < extendedImage.Width; i++)
         {
            for (int j = 0; j < extendedImage.Height; j++)
            {
               f[0][i, j] = extendedImage.GetPixel(i, j).R;
               f[1][i, j] = extendedImage.GetPixel(i, j).G;
               f[2][i, j] = extendedImage.GetPixel(i, j).B;
            }
         }
         var resultImage = new Bitmap(extendedImage);
         var g = f.AsParallel().Select(fi =>
         {
            return GaussianBlur(fi);
         }).ToArray();
         for (int i = 0; i < extendedImage.Width; i++)
         {
            for (int j = 0; j < extendedImage.Height; j++)
            {
      resultImage.SetPixel(i, j, Color.FromArgb((int)g[0][i, j], (int)g[1][i, j],
(int)g[2][i, j]));
            }
         }
         return ImageHelper.Crop(resultImage, initialImage, BlurSize);
      }
     public static double[,] GetCore()
      {
         var sigma = 3d;
         var sum = 0.0;
         var blurMatrix = new double[BlurSize, BlurSize];
```

```
                for (int l = 0; l < BlurSize; l++)
                {
                    for (int k = 0; k < BlurSize; k++)
                    {
blurMatrix[l, k] = 1 / Math.Sqrt(2 * Math.PI * sigma * sigma) * Math.Exp(-((l - BlurSize/2) * (l - BlurSize / 2)  + (k - BlurSize / 2) * (k - BlurSize / 2)) / (2 * sigma * sigma));
                        sum += blurMatrix[l, k];
                    }
                }
                for (int l = 0; l < BlurSize; l++)
                {
                    for (int k = 0; k < BlurSize; k++)
                    {
                        blurMatrix[l, k] /= sum;
                    }
                }
                return blurMatrix;
            }
            public static double[,] GaussianBlur(double[,] f)
            {
                var result = new double[f.GetLength(1), f.GetLength(0)];
                var blurMatrix = GetCore();
                for (int i = BlurSize / 2; i < f.GetLength(1) - BlurSize / 2; i++)
                {
                    for (int j = BlurSize / 2; j < f.GetLength(0) - BlurSize / 2; j++)
                    {
                        var temp = 0.0;
                        for (int l = -BlurSize / 2; l <= BlurSize / 2; l++)
                        {
                            for (int k = -BlurSize / 2; k <= BlurSize / 2; k++)
                            {
              temp += f[i - l, j - k] * blurMatrix[BlurSize / 2 + l, BlurSize / 2 + k];
                            }
                        }
                        result[i, j] = temp;
                    }
                }
                return result;
            }
        }
    }
    using System;
    using System.Collections.Generic;
    using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
 namespace ImageInterpolation.Filtering
{
   public static class MotionFilter
   {
      public static int MotionSize { get; set; }
      public static Direction Direction { get; set; }
           public static Bitmap Motion(Bitmap initialImage, int motionSize, Direction direction = Direction.LeftToRight)
      {
         MotionSize = motionSize;
         Direction = direction;
    var extendedImage = ImageHelper.GetExtended(initialImage, MotionSize);
         var f = new double[3][,];
         f[0] = new double[extendedImage.Width, extendedImage.Height];
         f[1] = new double[extendedImage.Width, extendedImage.Height];
         f[2] = new double[extendedImage.Width, extendedImage.Height];
         for (int i = 0; i < extendedImage.Width; i++)
         {
            for (int j = 0; j < extendedImage.Height; j++)
            {
               f[0][i, j] = extendedImage.GetPixel(i, j).R;
               f[1][i, j] = extendedImage.GetPixel(i, j).G;
               f[2][i, j] = extendedImage.GetPixel(i, j).B;
            }
         }
         var resultImage = new Bitmap(extendedImage);
         var g = f.AsParallel().Select(fi =>
         {
            return MotionBlur(fi);
         }).ToArray();
   for (int i = 0; i < extendedImage.Width; i++)
         {
            for (int j = 0; j < extendedImage.Height; j++)
            {
     resultImage.SetPixel(i, j, Color.FromArgb((int)g[0][i, j], (int)g[1][i, j], (int)g[2][i, j]));
            }
         } return ImageHelper.Crop(resultImage, initialImage, MotionSize);
      }
      public static double[,] GetCore()
      {
         var motion = 1;
```

```
            var sum = 0d;
            var motionMatrix = new double[MotionSize, MotionSize];
            for (int i = 0; i < MotionSize; i++)
            {
                for (int j = 0; j < MotionSize; j++)
                {
                    if ((i == j && Direction == Direction.LeftToRight)
          || (i == MotionSize - j - 1 && Direction == Direction.RightToLeft)
           || (j == MotionSize/2 && Direction == Direction.Horizontal)
            || (i == MotionSize / 2 && Direction == Direction.Vertical))
                    {
                        motionMatrix[i, j] = motion;
                    }
                    else
                    {
                        motionMatrix[i, j] = 0;
                    }
                    sum += motionMatrix[i, j];
                }
            }
            for (int l = 0; l < MotionSize; l++)
            {
                for (int k = 0; k < MotionSize; k++)
                {
                    motionMatrix[l, k] /= sum;
                }
            }
            return motionMatrix;
        }
        public static double[,] MotionBlur(double[,] f)
        {
            var result = new double[f.GetLength(1), f.GetLength(0)];
            var motionMatrix = GetCore();
            for (int i = MotionSize / 2; i < f.GetLength(1) - MotionSize / 2; i++)
            {
              for (int j = MotionSize / 2; j < f.GetLength(0) - MotionSize / 2; j++)
                {
                    var temp = 0.0;
                    for (int l = -MotionSize / 2; l <= MotionSize / 2; l++)
                    {
                        for (int k = -MotionSize / 2; k <= MotionSize / 2; k++)
                        {
temp += f[i - l, j - k] * motionMatrix[MotionSize / 2 + l, MotionSize / 2 + k];
                        }
                    }
```

```
                result[i, j] = temp > 255 ? 255 : temp;
                result[i, j] = result[i, j] < 0 ? 0 : result[i, j];
            }
        }
        return result;
    }
  }
  public enum Direction
  {
    RightToLeft,
    LeftToRight,
    Horizontal,
    Vertical
  }
}
using System;
using System.Drawing;
using System.Linq;
namespace ImageInterpolation.Filtering
{
  public static class SharpenFilter
  {
    public static int SharpSize { get; set; } = 3;
    public static double SharpPower { get; set; }

    public static Bitmap Sharpen(Bitmap initialImage, int sharpSize)
    {
      SharpPower = Convert.ToDouble(sharpSize);
   var extendedImage = ImageHelper.GetExtended(initialImage, SharpSize);
      var f = new double[3][,];
      f[0] = new double[extendedImage.Width, extendedImage.Height];
      f[1] = new double[extendedImage.Width, extendedImage.Height];
      f[2] = new double[extendedImage.Width, extendedImage.Height];
      for (int i = 0; i < extendedImage.Width; i++)
      {
        for (int j = 0; j < extendedImage.Height; j++)
        {
          f[0][i, j] = extendedImage.GetPixel(i, j).R;
          f[1][i, j] = extendedImage.GetPixel(i, j).G;
          f[2][i, j] = extendedImage.GetPixel(i, j).B;
        }
      }
      var resultImage = new Bitmap(extendedImage);
      var g = f.AsParallel().Select(fi =>
      {
```

```
                return Sharp(fi);
            }).ToArray();
            for (int i = 0; i < extendedImage.Width; i++)
            {
                for (int j = 0; j < extendedImage.Height; j++)
                {
                        resultImage.SetPixel(i, j, Color.FromArgb((int)g[0][i, j],
(int)g[1][i, j], (int)g[2][i, j]));
                }
            }
            return ImageHelper.Crop(resultImage, initialImage, SharpSize);
        }
        public static double[,] GetCore()
        {
            var sharp = SharpPower;
            var sharpMatrix = new double[SharpSize, SharpSize];
            for (int l = 0; l < SharpSize; l++)
            {
                for (int k = 0; k < SharpSize; k++)
                {
                    sharpMatrix[l, k] = l == k && k == 1 ? sharp : (1 - sharp) / 8;
                }
            }
            return sharpMatrix;
        }
        private static double[,] Sharp(double[,] f)
        {
            var result = new double[f.GetLength(0), f.GetLength(1)];
            var sharpMatrix = GetCore();
            for (int i = SharpSize / 2; i < f.GetLength(0) - SharpSize / 2; i++)
            {
                for (int j = SharpSize / 2; j < f.GetLength(1) - SharpSize / 2; j++)
                {
                    var temp = 0.0;
                    for (int l = -SharpSize / 2; l <= SharpSize / 2; l++)
                    {
                        for (int k = -SharpSize / 2; k <= SharpSize / 2; k++)
                        {
                                temp += f[i - l, j - k] * sharpMatrix[SharpSize / 2 + l,
SharpSize / 2 + k];
                        }
                    }
                    result[i, j] = temp > 255 ? 255 : temp;
                    result[i, j] = result[i, j] < 0 ? 0 : result[i, j];
                }
```

```
                }
                return result;
            }
        }
    }
    using Accord.Imaging;
    using System;
    using System.Drawing;
    using System.Numerics;
    namespace ImageInterpolation.Filtering
    {
       public static class WienerFilter
       {
        public static Bitmap Filter(Bitmap initialImage, ImageHelper.Filter filter)
          {
             var g = ComplexImage.FromBitmap(initialImage);
          var h = ImageHelper.GetComplexImageFromMatrix(GetCore(g, filter));
             var snr = ImageHelper.GetSNR(g.Data);

var G = ImageHelper. GetComplexImageFromMatrix(ImageHelper.FFT2
( ImageHelper.ToVector(g.Data)));
var H = ImageHelper.GetComplexImageFromMatrix(ImageHelper.FFT2
(ImageHelper.ToVector(h.Data)));

var F = GetF(H, G, 0.035);
var f = ImageHelper.GetComplexImageFromMatrix(ImageHelper.BFT2
(ImageHelper.ToVector(F.Data)));
    ImageHelper.Rotate(f);
    return f.ToBitmap();
    }
public static ComplexImage GetF(ComplexImage H, ComplexImage G, Complex
snr)
            {
                var bitmap = new Bitmap(G.Width, G.Height);
                var bitmap8bpp = bitmap.ConvertTo8bpp();
                bitmap8bpp.ConvertColor8bppToGrayscale8bpp();
                var complexImage = ComplexImage.FromBitmap(bitmap8bpp);
                for (int i = 0; i < G.Height; i++)
                {
                   for (int j = 0; j < G.Width; j++)
                   {
    complexImage.Data[i, j] = ((1 / H.Data[i, j]) *
(Math.Pow(Complex.Abs(H.Data[i, j]), 2)
             / (Math.Pow(Complex.Abs(H.Data[i, j]), 2) + snr))) * G.Data[i, j];
                   }
```

```
            }
            return complexImage;
        }
    public static double[,] GetCore(ComplexImage g, ImageHelper.Filter filter)
        {
            var matrixSize = 0;
            var matrix = new double[matrixSize, matrixSize];
            switch (filter)
            {
                case ImageHelper.Filter.Gauss:
                    matrixSize = GaussianFilter.BlurSize;
                    matrix = GaussianFilter.GetCore();
                    break;
                case ImageHelper.Filter.Sharpen:
                    matrixSize = SharpenFilter.SharpSize;
                    matrix = SharpenFilter.GetCore();
                    break;
                case ImageHelper.Filter.MotionLeftToRight:
                    matrixSize = MotionFilter.MotionSize;
                    MotionFilter.Direction = Direction.LeftToRight;
                    matrix = MotionFilter.GetCore();
                    break;
                case ImageHelper.Filter.MotionRightToLeft:
                    matrixSize = MotionFilter.MotionSize;
                    MotionFilter.Direction = Direction.RightToLeft;
                    matrix = MotionFilter.GetCore();
                    break;
                default:
                    break;
            }
            var result = new double[g.Width, g.Height];
            for (int l = 0; l < g.Height / 2 - matrixSize / 2 - 1; l++)
            {
                for (int k = 0; k < g.Width / 2 - matrixSize / 2 - 1; k++)
                {
                    result[l, k] = 0;
                }
            }
            for (int l = g.Height / 2 + matrixSize / 2; l < g.Height; l++)
            {
                for (int k = g.Width / 2 + matrixSize / 2; k < g.Width; k++)
                {
                    result[l, k] = 0;
                }
            }
```

```
for (int l = g.Height / 2 - matrixSize / 2 - 1; l < g.Height / 2 + matrixSize / 2; l++)
                {
 for (int k = g.Width / 2 - matrixSize / 2 - 1; k < g.Width / 2 + matrixSize / 2; k++)
                    {
        result[l, k] = matrix[l - (g.Height / 2 - matrixSize / 2 - 1), k - (g.Width / 2 -
matrixSize / 2 - 1)];
                    }
                }
                return result;
            }
        }
    }
    using Accord.Imaging;
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Linq;
    using System.Numerics;
    using System.Text;
    using System.Threading.Tasks;
    namespace ImageInterpolation.Filtering
    {
        class WienerPredictFilter
        {
            public static double[,] Core;
            public static Bitmap Filter(Bitmap initialImage, Bitmap broken)
            {
                var g = ComplexImage.FromBitmap(broken);
        var G = ImageHelper.GetComplexImageFromMatrix(ImageHelper.FFT2
(ImageHelper.ToVector(g.Data)));
                var snr = ImageHelper.GetSNR(g.Data);
                var filters = new ImageHelper.Filter[]
                {
                    ImageHelper.Filter.Gauss,
                    ImageHelper.Filter.MotionLeftToRight,
                    ImageHelper.Filter.MotionRightToLeft,
                };
                var bestImages = new Bitmap[filters.Length];
                var cores = new List<double[,]>();
                for (int i = 0; i < filters.Length; i++)
                {
                    var previousf = broken;
                    var nextf = broken;
                    var prevQuality = 0.0;
                    var nextQuality = 0.0;
```

```
                    var k = 0;
                    var matrixSize = 1;
                    var eps = 0.1;
                    var core = new double[matrixSize, matrixSize];
                    do
                    {
                        matrixSize += 2;
                        snr = GetSNR(previousf);
                        previousf = nextf;
                        core = GetCore(g, filters[i], matrixSize);
                        var h = ImageHelper.GetComplexImageFromMatrix(core);
        var H = ImageHelper.GetComplexImageFromMatrix(ImageHelper.FFT2
(ImageHelper.ToVector(h.Data)));
                        var F = WienerFilter.GetF(H, G, snr);
        var f = ImageHelper.GetComplexImageFromMatrix(ImageHelper.BFT2
(ImageHelper.ToVector(F.Data)));
                        ImageHelper.Rotate(f);
                        nextf = f.ToBitmap();
                        prevQuality = ImageHelper.GetPSNR(initialImage, previousf);
                        nextQuality = ImageHelper.GetPSNR(initialImage, nextf);
                        k++;
                    } while (nextQuality - prevQuality > eps || k < 2);
                    bestImages[i] = previousf;
                    cores.Add(core);
                }
                var psnrs = new List<double>();
                for (int i = 0; i < bestImages.Length; i++)
                {
                    psnrs.Add(ImageHelper.GetPSNR(initialImage, bestImages[i]));
                }
                var bestIndex = psnrs.IndexOf(psnrs.Max());
                Core = cores[bestIndex];
                return bestImages[bestIndex];
        public static double[,] GetCore(ComplexImage g, ImageHelper.Filter filter,
int matrixSize)
            {
                var matrix = new double[matrixSize, matrixSize];
                switch (filter)
                {
                    case ImageHelper.Filter.Gauss:
                        GaussianFilter.BlurSize = matrixSize;
                        matrix = GaussianFilter.GetCore();
                        break;
                    case ImageHelper.Filter.Sharpen:
                        SharpenFilter.SharpPower = matrixSize;
```

```
                        matrix = SharpenFilter.GetCore();
                        matrixSize = SharpenFilter.SharpSize;
                        break;
                    case ImageHelper.Filter.MotionLeftToRight:
                        MotionFilter.MotionSize = matrixSize;
                        MotionFilter.Direction = Direction.LeftToRight;
                        matrix = MotionFilter.GetCore();
                        break;
                    case ImageHelper.Filter.MotionRightToLeft:
                        MotionFilter.MotionSize = matrixSize;
                        MotionFilter.Direction = Direction.RightToLeft;
                        matrix = MotionFilter.GetCore();
                        break;
                    default:
                        break;
                }
                var result = new double[g.Width, g.Height];
                for (int l = 0; l < g.Height / 2 - matrixSize / 2 - 1; l++)
                {
                    for (int k = 0; k < g.Width / 2 - matrixSize / 2 - 1; k++)
                    {
                        result[l, k] = 0;
                    }
                }
                for (int l = g.Height / 2 + matrixSize / 2; l < g.Height; l++)
                {
                    for (int k = g.Width / 2 + matrixSize / 2; k < g.Width; k++)
                    {
                        result[l, k] = 0;

for (int l = g.Height / 2 - matrixSize / 2 - 1; l < g.Height / 2 + matrixSize / 2; l++)
                {
                     for (int k = g.Width / 2 - matrixSize / 2 - 1; k < g.Width / 2 +
matrixSize / 2; k++)
                    {
                            result[l, k] = matrix[l - (g.Height / 2 - matrixSize / 2 - 1), k -
(g.Width / 2 - matrixSize / 2 - 1)];
                    }
                }
                return result;
            }
        }
    }
```