ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ОЛЕСЯ ГОНЧАРА

Факультет прикладної математики

Кафедра обчислювальної математики та математичної кібернетики

**КУРСОВА РОБОТА**

за спеціальністю

на тему **«Назва теми»**

Другий (магістерський) рівень вищої освіти

Спеціальність 113 Прикладна математика

Освітня програма «Комп’ютерне моделювання та обчислювальні методи»

Виконавець

студент групи ПМ-20м-1

Сіряк Аліна Юріївна

Керівник

науковий ступінь, посада

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гарт Л.Л.

Кількість балів

Оцінка за національною шкалою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії:

\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_

2021

РЕФЕРАТ

*Курсова робота:* с., рис., табл., джерел, додатки.

*Об'єкт дослідження:* Наближені алгоритми розв’язання задач масштабування та відновлення 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 сформована інтегральна постановка задачі. 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 досліжденнях, у астрономії, у машинному навчанні та в задачах розпізнавання. Також отримані результати можна використати у процесі оптимізації та покращення роботи існуючих алгоритмів відновлення, а також для розширення можливостей їх застосування.

*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](#__RefHeading___Toc359252463) |  |
| ПОСТАНОВКА ЗАДАЧІ |  |
| 1 АЛГОРИТМИ ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ |  |
| 1.1 Метод регуляризації Тихонова |  |
| 1.2 Метод регуляризації Фрідмана |  |
| 1.3 Фільтр Вінера |  |
| 2 ПРОГРАМНА РЕАЛІЗАЦІЯ |  |
| 2.1 Опис програми |  |
| 2.2 Опис розрахункових експериментів |  |
| 2.3 Порівняльна характеристика алгоритмів |  |
| [ВИСНОВКИ](#__RefHeading___Toc359252484) |  |
| ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ |  |
| ДОДАТОК Лістинг програми |  |

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.

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.

Розподіл основних видів обробки зображень:

1. 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.

1. 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.

1. 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.

1. 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. 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 описуюється 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 спотворенного зображення описується набором одновимірних інтегральних рівнянь Фредгольма I роду типу згортки

або двовимірним інтегральним рівнянням Фредгольма I роду типу згортки:

де – ядро інтегрального рівняння, що являє собою функцію розсіювання точки, у більшості випадків просторово-інваріантної; – лінійний інтегральний оператор;і– розподіл інтенсивності за істинним та спотвореним зображеннями відповідно; – 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].

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 Згорткові фільтри

Загальні теоретичні відомості

Сформулюємо постановку задачі у термінах згортки.

Введемо множину .

Нехай спотворене зображення позначено як *,* де — ширина зображення, — висота зображення, 255 — кількість можливих кольорів пікселів.

Оригінальне зображення позначено як .

Функція спотворення визначена , де - ширина матриці ядра спотворення, - висота матриці ядра спотворення. Мають виконуватися наступні умови: *, , , , .*

Функція адитивного шуму визначена як .

Функція нормування позначена .

, - координати ширини та висоти пікселя у зображенні.

Тоді модель процесу спотворення зображення можна описати таким чином:

Функція з формули (1.1), що спотворює зображення, зазвичай являє собою цифровий фільтр, що був застосований до зображення. Цифрові фільтри дозволяють накладати на зображення різні ефекти, такі як розмиття, підвищення різкості, деформація, підвищення шуму, ерозія, нарощення, зміна контрастності та кольорового спектру та багато інших.

Цифрові фільтри являють собою алгоритми обробки зображення. Більшість таких фільтрів використовують схожі алгоритми, проте ефекти цих фільтрів визначаються коефіцієнтами, що застосовуються в алгоритмі. Навіть невелика зміна в одному коефіцієнті може призвести до значних змін у вихідному результаті, тому правильне та точне визначення коефіцієнтів алгоритму є важливою задачею.

Згортка є способом представлення будь-якого векторного значення скалярним значенням [3, ст. 406]. Існує безліч способів такого представлення, але стосовно обробки зображень векторне значення являє собою колір групи пікселів, а скалярне значення, що отримується на основі згортки, являє собою колір пікселя, отриманого в результаті застосування фільтру до вхідного зображення. При використанні згорткових фільтрів згорткою є перетворення матриці RGB -каналів пікселів вхідного у прямокутних координатах зображення з використанням іншої матриці, що є матрицею коефіцієнтів цифрового фільтру та має назву “ядро”.

Цифрові фільтри на основі згортки характеризуються розміром групи пікселів та своєю імпульсною характеристикою.

Перетворення із застосуванням матриці коефіцієнтів відбувається наступним чином. Кожен елемент вихідної матриці множиться на центральне значення матриці ядра. Крім цього на відповідні значення множаться навколишні його елементи, після чого результати сумуються і приймаються як перетворене значення.

Далі вираховується сума, де складовими є добутки значень пікселів на значення комірки ядра, що оточує даний піксель. Визначена сума є коефіцієнтом нормування, застосування якого є необхідним для збереження середньої інтенсивності зображення. Сума ділиться на суму всіх елементів ядра згортки. Отримане значення як раз і є новим значенням обраного пікселя. Якщо застосувати згортку до кожного пікселя зображення, то в результаті вийде ефект, що залежить від обраного ядра згортки.

Обробка зображення із застосуванням такого роду фільтрів описується формулою (2.1).

|  | (2.1) |
| --- | --- |

де - відфільтроване зображення, — оригінальне зображення, — ядро згортки. Кожен елемент ядра згортки обмежений та .

Застосування матриці згортки для центрального пікселя відбувається наступним чином:

= (12 \* 0.5 + 14 \* 0.75 + 41 \* 0.5 + 43 \* 0.75 + 84 \* 1.0 + 24 \* 0.75 + 2 \* 0.5 + 1 \* 0.75 + 43 \* 0.5) \*= 32. 41667

Основним завданням при розробці цифрових фільтрів з кінцевою імпульсною характеристикою є розрахунок коефіцієнтів фільтра. Розглянемо типові приклади фільтрів з кінцевою імпульсною характеристикою та матриці їх коефіцієнтів.

Фільтр Гауса

Фільтри розмиття є найбільш часто застосованою категорією згорткових фільтрів. Фільтр Гауса належить до цієї категорії та виконує найякісніше розмиття.

Ідея фільтру полягає у використанні нормального розподілу (також відомого як Гаусове) для обчислення перетворення, що застосовується до кожного пікселя зображення.

Формула нормального розподілу:

де – значення, отримане після розрахунку формули Гауса для пікселя з координатами ; – ступінь бажаного розмиття зображення;

– координати пікселя зображення по горизонталі та вертикалі відповідно;

Реалізація формули припускає, що значення і дорівнюють нулю при представленні координат пікселя, розташованого в середині ядра. При обчисленні елементів ядра значення координат, виражені і , повинні відображати відстань в пікселях від середнього пікселя.

Застосування формули Гаусового розподілу для визначення згортки [1]:

Імпульсна характеристика для отриманого ядра виглядає наступним чином:



Рисунок 2.5. Зображення ядра Гауса

Розмір матриці впливає на силу розмиття:



Рисунок 2.6. а) оригінальне зображення б) розмиття матрицею 5х5 в) розмиття матрицею 10х10

Фільтр зсуву

Розмиття руху часто можна спостерігати в тих випадках, коли швидкі рухи фіксуються на фотографіях або відеозаписах. При записі одного кадру швидкі переміщення можуть привести до зміни зображення до завершення захоплення кадру [14].

Зсув можна штучно імітувати за допомогою фільтрів цифрового зсуву зображення. Розмір ядра матриці, що надається при реалізації згортки зображень, впливає на інтенсивність фільтра, використаного в результуючих зображеннях. Щодо фільтрів зсуву розмір ядра впливає на швидкість, з якою стався рух у певному напрямку та який призвів до розмиття. Великі ядра створюють видимість більш швидкого руху, тоді як більш дрібні ядра призводять до сприйняття менш швидкого руху.

Залежно від зазначеного ядра існує можливість створювати видимість руху, що стався в певному напрямку.

Коефіцієнт нормалізації визначається як обернена сума всіх елементів у матриці. Елементи матриці завжди дорівнюють 0 або 1, порядок їх розташування у матриці визначає напрямок руху.

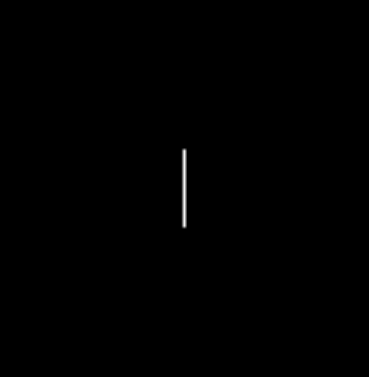
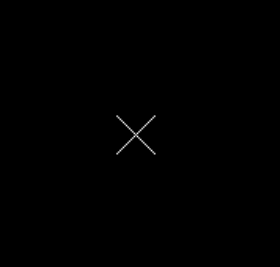
  

Рисунок 2.8 Приклад зображення ядер змазу

1.2 Фільтр Вінера

Розглянемо модель спотворення зображення (1.1).

Для того, щоб розв'язати задачу відновлення, необхідно з формули(1.1) знайти . Вирішення цієї задачі може призвести до розв'язування великої системи рівнянь, що не є зручним способом знаходження та значно сповільнює виконання програми.

Для подолання цієї проблеми використовують перетворення Фур'є [3, cт. 284]. 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:

Для перетворення Фур'є функцій заданих на просторі використовується формула багатовимірного інтегрального перетворення:  
   
 Зворотне перетворення у такому випадку задається формулою:  
   
 У деяких джерелах [5, 12] відрізняється перед інтегральний коефіцієнт для багатовимірного перетворення, що може незначно вплинути на вихідний результат.

Зазвичай для обробки сигналів (наприклад, у стисканні звука в МР3 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:

|  | (2.2) |
| --- | --- |

Аналогічна формула для зворотного перетворення:

|  | (2.3) |
| --- | --- |

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 та використаємо формули для прямого та зворотного дискретного перетворення Фур'є:

де *FFT* – операція прямого перетворення Фур'є (Forward Fourier Transform) (2.2), *BFT* – операція зворотного перетворення Фур'є (Backward Fourier Transform) (2.3)

Використавши формулу прямого перетворення Фур'є, перепишемо задачу (1.1) таким чином:

|  | (2.4) |
| --- | --- |

де – функції після прямого перетворення Фур'є відповідно, — координати ширини та висоти відповідно.

Найпростіший спосіб приблизно знайти – розділити рівняння (2.4) на .

Отримаємо таку оцінку оригінального зображення:

Проте такий спосіб майже не використовується на практиці та дає погані результати. Якщо функція приймає близькі до нуля значення (що майже завжди так), то вклад останнього доданку буде домінуючим. Щоб уникнути цього ефекту, значення функції теж мають бути максимально близькими до нуля. Тому додання невеликого шуму до зображення призводить до значних візуальних недоліків [3, ст. 411, 7].

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 [1, 7]. Алгоритм мінімізує квадратичну похибку між початковим та відновленим зображенням.

Мета роботи фільтру полягає у пошуку такої оцінки для оригінального зображення , щоб середньоквадратичне відхилення цих зображень було мінімальне.

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

де – математичне сподівання.

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

,

де та — енергетичні спектри оригінального та відновленого зображення.

Спектральна щільність сигналу визначається відношенням:

де– автокореляційна функція

Взаємна спектральна щільність сигналу:

де – функція взаємної кореляції

Відновлення зображення відбувається з використанням наступного відновлюючого перетворювача:

Знайдемо наближене відновлене зображення за формулою:

|  | (2.5) |
| --- | --- |

Параметри являють собою енергетичні спектри шуму та зображення відповідно. Знаходження цих параметрів є окремою підзадачею. Існує декілька способів їх визначення.

Відношення можна приблизно визначити як відношення сигналу до шуму (Signal-to-Noise Ratio, SNR) [4, 6].

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



Рисунок 2.9 Сигнал значно більший за шум

У наведеному вище зображенні (рис. 2.9) 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. 2.10).

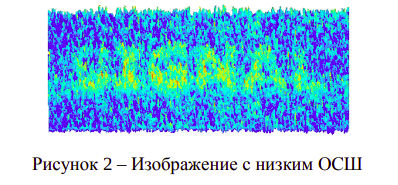


Рисунок 2.10 Сигнал подібний до шуму

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

|  | (2.1) |
| --- | --- |

де − середнє значення яскравості; − стандартне відхилення.

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

Існує альтернативне визначення коефіцієнтів та[7]:

|  | (2.2) |
| --- | --- |

де — математичне сподівання, — оригінальне зображення, — 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:

Для оцінки середньоквадратичного відхилення використовуємо формулу:

де – високочастотна область, що розглядається в даний момент, – спектр спотвореного зображення.

Щоб застосувати фільтр Вінера до зображення, потрібно:

1. Перевести матрицю цілих чисел зображення у матрицю комплексних чисел.

2. Визначити спотворюючу функцію у вигляді дискретного набору значень.

3. Застосувати пряме перетворення Фур'є до матриці зображення та матриці спотворюочої функції за формулою (2.2), отримавши матриці та .

4. Розрахувати відношення як відношення сигналу до шуму (SNR) за формулою (2.6), або окремо розрахувати та за формулами(2.7).

5. Розрахувати наближене відновлене зображення за формулою (2.5).

6. Застосувати зворотне перетворення Фур'є за формулою (2.3) до зображення , отримавши вихідне зображення .

1.3 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 (ξ,η) і А - оператор, який впливає на об'єкт, тоді можна визначити вплив деякого оператора на об'єкт: .

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

, де — зображення, яке максимально наближене до об'єкта.

Метод формування зображення проілюстровано на рис. 2.1.

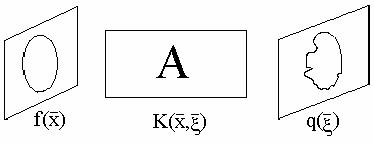


Рисунок 2.1 - Одноступінчатий метод формування зображення

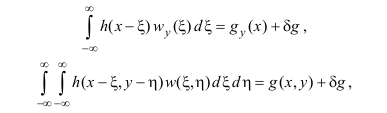
Як правило, такий процес формування зображення описується оператором в інтегральної формі:



де  – 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 сверхразрешенія.

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



Для вирішення інтегральних рівнянь (1) і (2) найбільш підходить метод регуляризації Тихонова [2-7, 12]. 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, 12].

Метод регуляризації Тихонова з використанням перетворення Фур'є застосуємо для рішення рівнянь типу згортки. Розглянемо окремий випадок ІУ Фредгольма I роду - рівняння типу згортки одномірне (1) і двовимірне (2). Якщо ІУ 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 (1) щодо wy (ξ) при кожному фіксованому значенні y, що грає роль параметра. У рівнянні (1) ФРТ виражається формулою [2, 3, 6]:



де ∆ — величина змазу.

Просторова инвариантность функції h має місце в разі, коли величина ∆ не залежить від x (але може залежати від y, в цьому, досить рідкісному, випадку ∆ = ∆y, h = hy). Якщо значення ∆ 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:



де ФРТ визначається як:

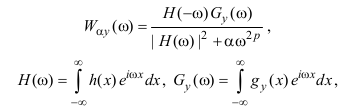


Як у випадку рівняння (1), так і (3), необхідно вирішити стільки самостійних (необразующіх систему) одновимірних рівнянь, скільки рядків містить зображення.

Рішення рівняння (1) методом ПФ з регуляризації Тихонова має вигляд [2, 3, 5-7, 12]:



де



- перетворення Фур'є від g y (x) і h (x) відповідно, α> 0 - параметр регуляризації, а p ≥ 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 (3) і (1) 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:

,

де — матриця розмірності , що пов'язана з ФРТ (4), тобто , де - дискретні цілочисельні відліки за пікселами вздовж -го рядка ( - число стовбців в зображенні). Зазначимо, що вектори та мають довжину .

У методі регуляризації Тихонова замість (3) вирішується рівняння

,

де α > 0 — параметр регуляризації, E — одиничний оператор (одинична матриця). Матриця є квадратною, симетричною і позитивно-визначеною, СЛАР (4) має рішення і чисельно стійка [2, 3, 12, 14]. В результаті отримуємо рішення у вигляді:

Для того щоб реалізувати метод регляризації Тихонова необхідно:

1. Визначити параметр регуляризації: за емпіричними даними або методами наведеними у ().
2. Застосувавши метод квадратур, розрахувати матрицю А за формулою ().
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.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Опис програми

Постановка задачі

Нехай на вхід програми задане цифрове зображення розміром . Величини та – розміри піксельної сітки, на елементах якої задані значення , що характеризують яскравість або колір пікселя у зображенні, що було зіпсоване функцією . Задачею відновлення зображення є отримання відновленого зображення з відомого спотвореного зображення та даних про функцію спотворення з урахуванням функції шуму .

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 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.

Згорткові фільтри

Фільтр Гауса

Для розробки згорткових фільтрів було створено клас GaussianFilter, у якому знаходиться основний функціонал програми для створення гаусового розмиття. У ньому міститься параметр BlurSize, який визначає розмір розмиваючого ядра. Основний метод має назву Blur та приймає на вхід оригінальне зображення та розмір ядра розмиття. В цьому методі на початку відбувається побудова збільшеного на розмір цього ядра зображення на основі вхідного шляхом послідовного копіювання кутових пікселів у незаповнену ділянку для того, щоб фільтр був застосований до всього зображення. Із отриманого збільшеного зображення відбувається формування трьох наборів даних у вигляді матриці для кожного кольору спектру RGB. Для кожного такого набору даних відбувається застосування ядра у методі GaussianBlur.

Спочатку встановлюється параметр *σ* та у методі GetCore розраховується ядро Гауса за формулою:

Отримані коефіцієнти сумуються, після завершення формування матриці коефіцієнтів кожен з них розділяється на розраховану суму. Таким чином відбувається нормалізація ядра.

Після розрахунку матриці ядра необхідно провести його застосування до кожного пікселя зображення.

У основному методі відбувається застосування ядра до кожного набору даних спектру RGB. З отриманих наборів даних конструюється нове зображення. Перед завершенням роботи метода у вихідному зображенні відсікаються раніше додані пікселі, таким чином зображення набуває початкового розміру.   
Фільтр зсуву

Фільтр зсуву є подібним до вищезазначених фільтрів обробки зображень, окрім того що у класі MotionFilter, який виконує роботу по застосуванню фільтра, окрім параметру MotionSize для визначення розміру матриці є ще один параметр — MotionDirection, який визначає напрям розмиття.

У методі GetCore задається конкретний напрям зсуву та його сила. Залежно від обраного напряму, відповідні комірки матриці заповнюються 1 та 0. Одиницями заповнюються середні комірки матриці вздовж напряму зсуву. Інші комірки заповнюються нулями. Також необхідно провести нормалізацію отриманих значень та порахувати їх суму, після чого розділити кожне значення на отриману суму. Перед застосуванням матриці до зображення необхідно перевірити отримані значення на належність до проміжку [0, 255]. У разі невиконання цієї умови необхідно провести нормалізацію та привести наявні значення до значень цього проміжку.

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:

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:

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.

Необхідним для подолання небажаного шуму є розрахунок відношення сигналу до шуму. 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 (2.5) у методі GetF. Дані вихідного зображення отримуються розрахуванням цієї формули для кожного пікселя, та переведенням їх у формат комплексного зображення.

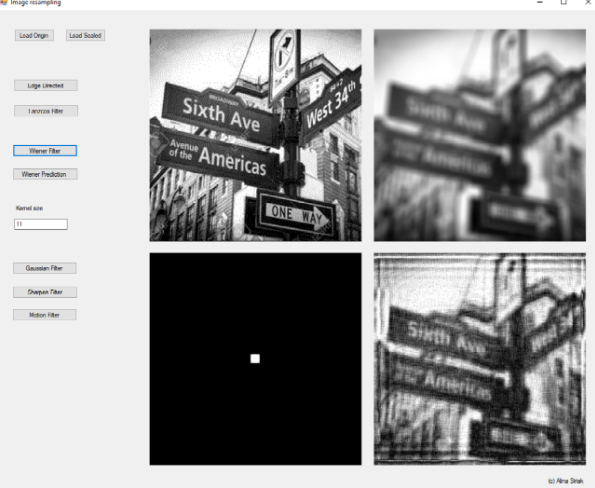
Після розрахунку оцінки оригінального зображення її необхідно перевести зворотною трансформацією Фур'є. Для цього, як і раніше, переведемо дані з матричного вигляду до векторного методом ToVector, розрахуємо методом BFT2 зворотну трансформацію Фур'є та перетворимо отримані дані до формату комплексного 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.

Метод регуляризації Тихонова

3.2 Опис розрахункових експериментів

Фільтр Вінера

Застосуємо фільтр Вінера до зображень із розмиттям та зсувом.



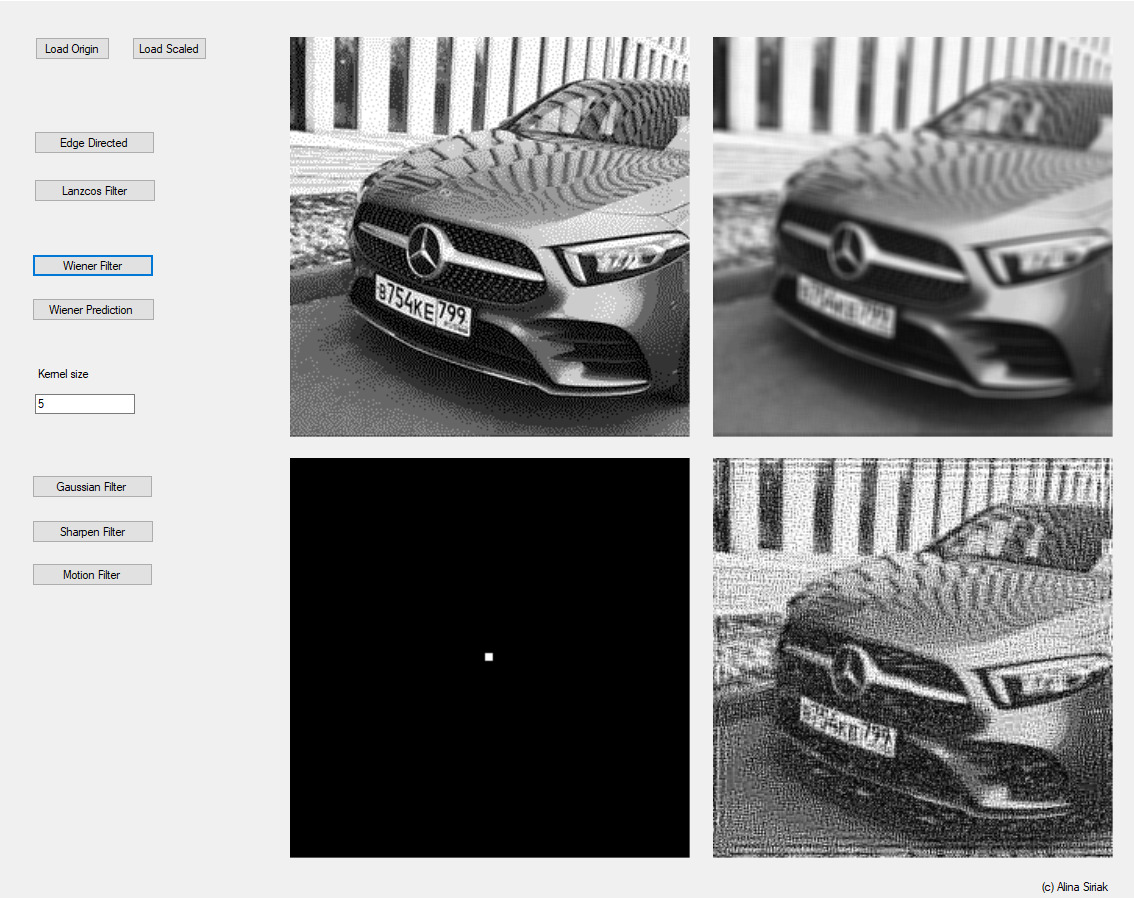
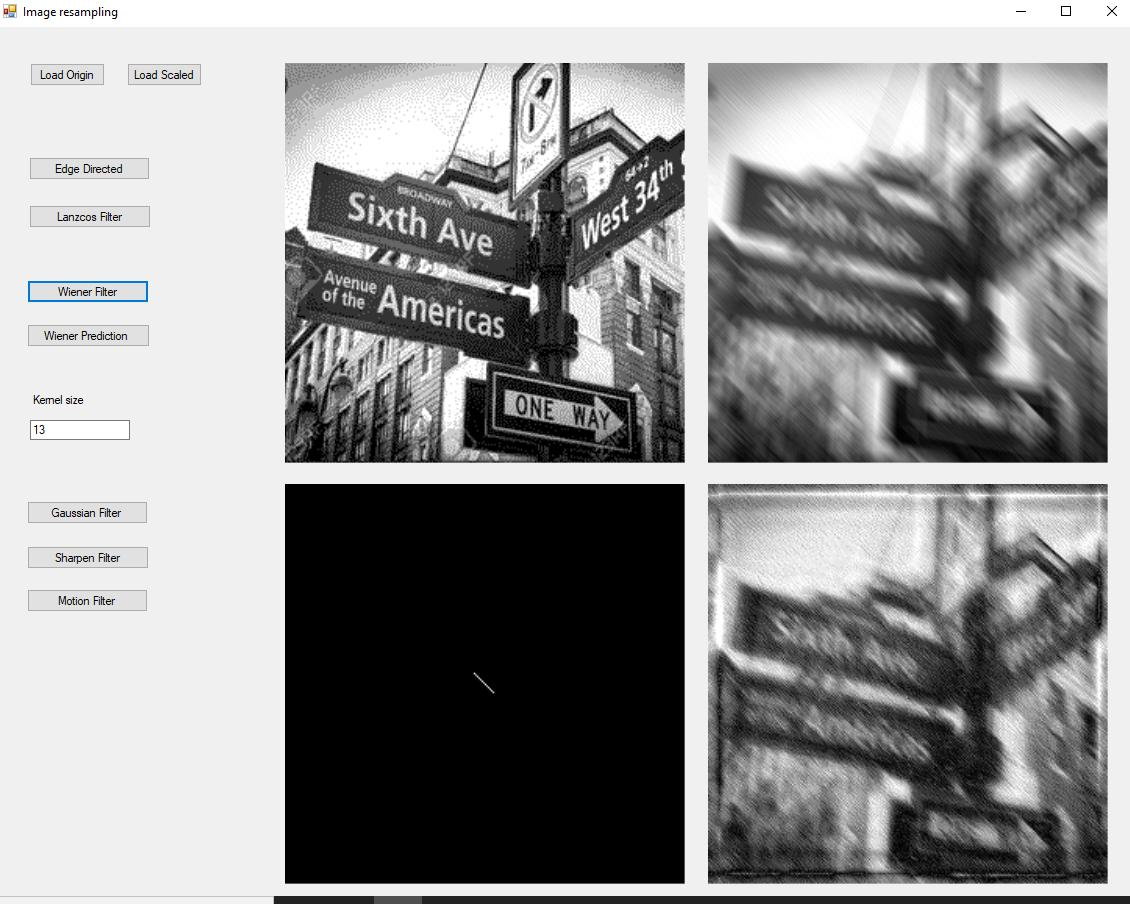


Рисунок 3.3 Розмиття з Гаусом та застосування фільтру Вінера



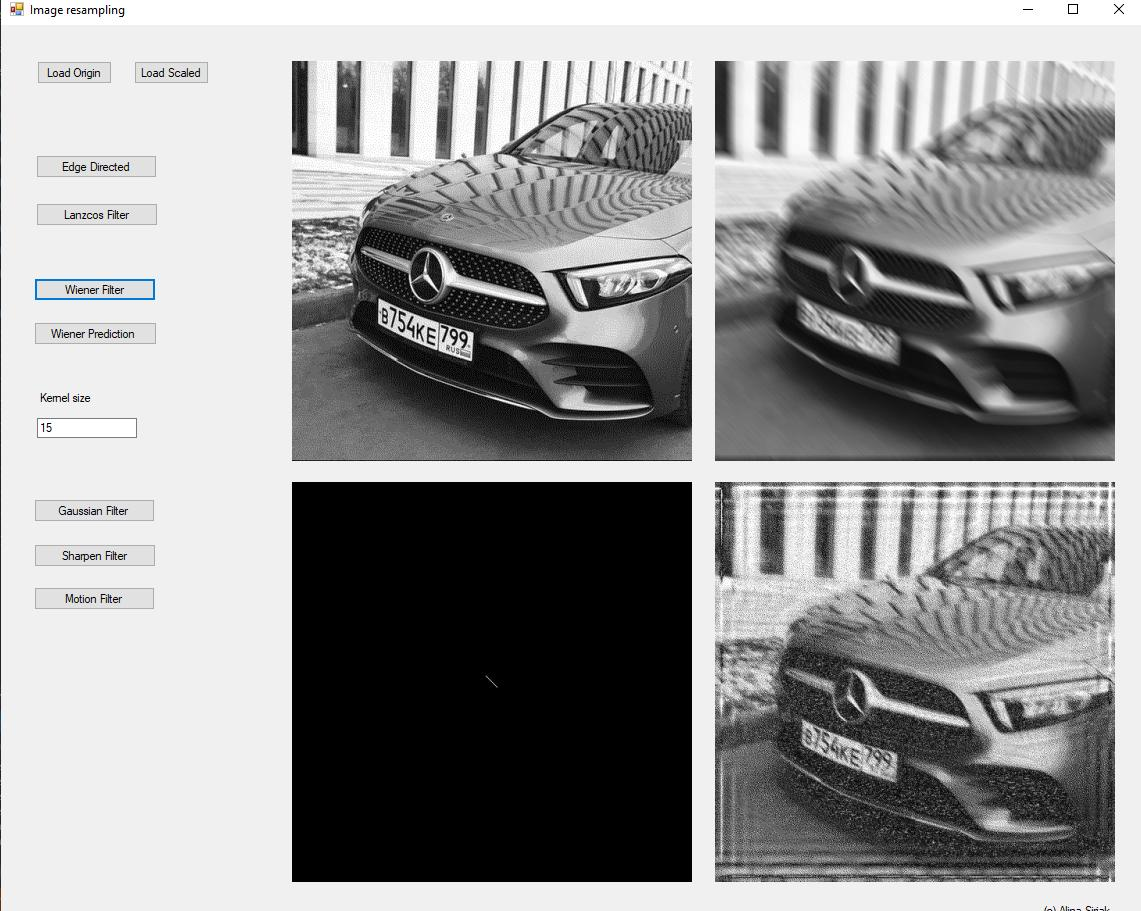


Рисунок 3.4 Змаз по діагоналі та застосування фільтру Вінера

3.3 Порівняльна характеристика алгоритмів

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

Порівняємо алгоритми простого покращення чіткості, фільтрації Вінера та фільтрації Вінера з невідомим ядром.

Швидкість роботи оцінена для зображень розмірів із застосуванням ядра розміром mxm та наведена у секундах.

Таблиця 3.1 Швидкість роботи алгоритмів для вхідних даних різної розмірності

*m* = 5:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Алгоритм покращення чіткості | Фільтр Вінера | Фільтр Вінера з невідомим ядром |
| *n* = 128 | 0.1805 | 0.3823 | 1.8783 |
| *n* = 256 | 0.5207 | 1.1925 | 5.5211 |
| *n* = 512 | 1.0334 | 2.5296 | 12.3873 |

*m* = 15*:*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Алгоритм покращення чіткості | Фільтр Вінера | Фільтр Вінера з невідомим ядром |
| *n* = 128 | 0.1943 | 0.3911 | 2.6562 |
| *n* = 256 | 0.5583 | 1.2129 | 7.7439 |
| *n* = 512 | 1.2534 | 2.5888 | 16.3783 |

*m* = 25:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Алгоритм покращення чіткості | Фільтр Вінера | Фільтр Вінера з невідомим ядром |
| *n* = 128 | 0.1998 | 0.3971 | 4.0258 |
| *n* = 256 | 0.6025 | 1.2247 | 10.0349 |
| *n* = 512 | 1.4712 | 2.6981 | 23.3730 |

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.

Розрахуваши PSNR для вихідних зображень розміром ,= 256, маємо такі результати:

Таблиця 3.2 Оцінка якості отриманих результатів для різних типів спотворень

Розмиття:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Алгоритм покращення чіткості | Фільтр Вінера | Фільтр Вінера з невідомим ядром |
| *m* = 5 | 20.32 | 21.83 | 21.61 |
| *m* = 15 | 19.91 | 20.99 | 21.09 |
| *m* = 25 | 18.48 | 19.36 | 18.97 |

Зсув:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Алгоритм покращення чіткості | Фільтр Вінера | Фільтр Вінера з невідомим ядром |
| *m* = 5 | 17.83 | 18.85 | 18.48 |
| *m* = 15 | 15.61 | 17.02 | 15.85 |
| *m* = 25 | 13.63 | 16.53 | 14.04 |

Візуальне порівняння.

Результат роботи алгоритмів наведений на рис.3.7 та рис.3.8.



а)

 б) в) г)

Рисунок 3.7. а) розмите зображення б) відновлене фільтром покращення чіткості зображення в) відновлене фільтром Вінера зображення г) відновлене фільтром Вінера з невідомим ядром зображення



а)



б) в) г)

Рисунок 3.8. а) змазане зображення б) відновлене фільтром покращення чіткості зображення в) відновлене фільтром Вінера зображення г) відновлене фільтром Вінера з невідомим ядром зображення

Алгоритм покращення чіткості виконується швидше ніж фільтр Вінера через відсутність додаткових операцій таких як виконання трансформацій Фур'є. У разі розмиття контури стають чіткішими, проте майже не вдається відновити дрібні деталі. Якість зображення менша за якість, що розрахована для фільтра Вінера, проте у разі невеликих спотворень розмиттям є задовільною. У разі зсуву алгоритм покращення чіткості дає поганий результат: контури стають чіткішими, проте помітне їх подвоєння. У разі значного зсуву алгоритм зовсім не дає бажаного ефекту, якість зображення дуже мала у зрівнянні з фільтром Вінера.

Фільтр Вінера працює довше, 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.

Робота фільтру Вінера з невідомим ядром займає багато часу, оскільки відбувається ітераційне виконання класичного фільтру Вінера. Із ростом сили спотворення значно зростає час виконання алгоритму. У разі розмиття будь-якої потужності вдається досить точно підібрати оптимальні параметри ядра, використання яких приводить до успішного результату. Отримане зображення дуже схоже на зображення отримане класичним алгоритмом. У разі незначного змазу алгоритм дає задовільний результат, але зі збільшенням змазу алгоритм перестає бути ефективним. Причина такої поведінки лежить у особливостях застосованої метрики якості зображення. PSNR є еталонною об’єктивною метрикою [9, 10], яка враховує степінь розмиття та наявність шуму на зображенні, проте не може правильно оцінити зображення зі змазом. Для подолання цієї проблеми необхідно розглядати інші способи оцінки зображень.

ВИСНОВКИ

При написанні даної дипломної роботи були проаналізовані різні методи відновлення зображень та детально описані алгоритми їх роботи. Був створений програмний продукт, який показав роботу таких алгоритмів, як алгоритм покращення чіткості, фільтр Вінера та фільтр Вінера з невідомим ядром, наведений опис цього продукту, а також представлена порівняльна характеристика результатів.

З отриманих результатів можна зробити наступні висновки:

* Алгоритм покращення чіткості виконується швидко, проте результати його роботи у багатьох випадках не є задовільними.
* Фільтр Вінера працює довше, але швидкість є задовільною. Він 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.
* Робота фільтру Вінера з невідомим ядром займає багато часу, із ростом сили спотворення значно зростає час виконання алгоритму. У разі розмиття будь-якої потужності вдається досить точно підібрати оптимальні параметри ядра, використання яких приводить до дуже успішного результату. У разі незначного змазу алгоритм дає задовільний результат, але зі збільшенням змазу алгоритм перестає бути ефективним. Причина такої поведінки лежить у особливостях застосованої метрики якості зображення. PSNR є еталонною об’єктивною метрикою, яка враховує степінь розмиття та наявність шуму на зображенні, проте не може правильно оцінити зображення зі змазом. Для можливості використання алгоритму у випадку недоліків типу змазу необхідно розглядати альтернативні способи оцінки зображень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бондина Н.М., Калмичков О.С., Кривенцов В.Е. Сравнительный анализ алгоритмов фильтрации медицинских изображений / УДК 004.932.4 // Вісник НТУ "ХПІ", Інформатика та моделювання, Харків НТУ "ХПІ", 2012 № 38. – С. 14 – 25. URL: <https://cyberleninka.ru/article/n/sravnitelnyy-analiz-algoritmov-filtratsii-meditsinskih-izobrazheniy/viewer>.

2. Гарт Л.Л., Сіряк А.Ю., Балейко Н.В. Про Реалізацію Алгоритмів Багатовимірної Апроксимації Функцій // Комп'ютерні науки, інформаційні технології та системи управління: Матеріали міжнародної науково-технічної конференції CSYSC здобувачів вищої освіти та молодих вчених – Івано-Франківськ, 2019. – C. 13-14.

3. Гонсалес Р., Вудс Р. Цифровая обработка изображений, 2012, 1104 с.

4. Голиков Е. Н. Измерение характеристик шумов и отношения сигнал/шум в телевизионных изображениях с использованием программных средств, Томск, 2012, 4 с.

5. Гудмен Дж. Введение в Фурье-оптику, Москва, 1970, 364 с.

6. Конюхов А.Л., Костевич А.Г., Курячий М.И. Критерии оценки отношения сигнал/шум в активноимпульсных телевизионно-вычислительных системах, УДК 621.397.13, Доклады ТУСУРа, № 2 (26), часть 1, декабрь 2012 — С.111-115. URL: <http://old.tusur.ru/filearchive/reports-magazine/2012-26-1/111.pdf>

7. Панфилова К.В. “Компенсация линейного смаза цифровых изображений с помощью метода Люси-Ричардсона“, Москва, 2015 г, GraphiCon2015 — С.163-167 URL:<https://gc2011.graphicon.ru/html/2015/papers/33.pdf>.

8. Переславцева Е.Е., Филлипов М.В. Метод ускоренного восстановления изображений, УДК 621.395, ISSN 1994-0408, 2012г., 13 с. URL: <http://technomag.edu.ru/doc/340562.html>.

9. Сидоров Д.В., Осокин А.Н., Марков Н.Г. Оценка качества изображений с использованием вейвлетов УДК 004.932.2, Известия Томского политехнического университета. 2009. Т. 315. № 5, - C.104-107. URL: <https://core.ac.uk/download/pdf/53067002.pdf>.

10. Шелухин О.И., Марков М.В. Сравнительный анализ метрик оценки качества восприятия потоковой информации, УДК 681.3.07, Электротехнические и информационные комплексы и системы №3, 2010 — С.43-47.

11. Biggs D.S.C., Andrews M. Acceleration of iterative image restoration algorithms, Applied Optics, 1997. – Vol. 36 (No. 8).

12. Goodman J. Introduction to Fourier Optics, McGraw Hill, 1996.

13. Mariana S.C. Almeida, Luis B. Almeida. Blind and Semi-Blind Deblurring of Natural Images, IEEE Transaction on Image processing, Vol 19, No.1, January 2010, p.36-52.

14. Michal Sorel and Jan Flusser, Space-Variant Restoration of Images Degraded by Camera Motion Blur, IEEE Transaction on image processing, Vol 17, No. 2 Febreary 2008, p.105-116.

15. Srinivasa Rao D., Selvani Deepthi K., Moni Sushma Deep K., Application of Blind Deconvolution Algorithm for Image Restoration, Faculty of Computer Science Applications, ISSN : 0975-5462 Vol. 3 No. 3, March 2011, 7 p. URL: [https://www.oalib.com/paper/2104736#.Xt-3U88zY5k](https://www.oalib.com/paper/2104736" \l ".Xt-3U88zY5k).

16. Young I.T., Vliet L.J.,[Recursive implementation of the Gaussian filter](https://d1rkab7tlqy5f1.cloudfront.net/TNW/Over faculteit/Decaan/Publications/1995/SP95TYLV.pdf), Signal Processing, vol. 44, no. 2, 1995, 13 p. URL: <https://www.tudelft.nl/en/faculty-of-applied-sciences/about-faculty/dean/lucas-van-vliet/publications/>.

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;

}

}

}