МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра спеціалізованих комп’ютерних систем

ІЛЬЧИШИН Михайло Ігорович

ПРОГРАМНИЙ МОДУЛЬ КОМП’ЮТЕРНО-ІНТЕГРОВАНОЇ СИСТЕМИ ВІДНОВЛЕННЯ НЕЧІТКИХ ЗОБРАЖЕНЬ / SOFTWARE MODULE OF THE COMPUTER-INTEGRATED SYSTEM FOR THE RESTORATION OF FUZZY IMAGE.

спеціальність: 151 – Автоматизація та комп’ютерно-інтегровані технології
магістерська програма – Автоматизація та комп’ютерно-інтегровані технології

Магістерська робота

Виконав: студент групи АКІТм-21
М. І. Ільчишин
____
Науковий керівник:
д.т.н., професор Я. М. Николайчук
____

Магістерську роботу допущено до захисту:
"____"____________20___ р.

Завідувач кафедри
_________________ Я.М. Николайчук

Тернопіль 2018

Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра спеціалізованих комп'ютерних систем
Освітній ступінь "магістр"
спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
магістерська програма – Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ
Завідувач кафедри СКС
______________________ Я.М.Николайчук
"____"______________20__ р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ
ІЛЬЧИШИН Михайло Ігорович

(прізвище, ім'я по-батькові)

1. Тема магістерської роботи

Програмний модуль комп'ютерно-інтегрованої системи відновлення нечітких зображень / software module of the computer-integrated system for the restoration of fuzzy image.

керівник роботи д.т.н., професор Я. М. Николайчук

затверджені наказом по університету від "14" листопада 2017 р. № 804

2. Строк подання студентом закінченої магістерської роботи 10 грудня 2018р.

3. Вихідні дані до магістерської роботи:

1. Розмите зображення
2. Методи реконволюції зображень
3. Методи визначення ядра розмиття
4. Методи фільтрації зображень

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)**:**

1. Математичні основи відновлення зображень.
2. Покращена методика відновлення розмитих зображень.
3. Вибір та налаштування інструментів для розробки системи відновлення зображень.
4. Розробка спеціалізованої комп'ютерної системи для відновлення нечітких зображень.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)**:**

____
____
____

____

____

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 14 листопада 2017 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Математичні основи відновлення зображень	11.2017р. – 01.2018р.	
2	Покращена методика відновлення розмитих зображень	02.2018р. – 04.2018р.	
3	Вибір та налаштування інструментів для розробки системи відновлення зображень	05.2018р. – 07.2018р.	
4	Розробка спеціалізованої комп’ютерної системи для відновлення нечітких зображень	08.2018р. – 10.12.2018р.	

Студент __________ М. І. Ільчишин
(підпис)

Керівник магістерської роботи ___________ д.т.н., професор Я. М. Николайчук
(підпис)

РЕФЕРАТ

Робота виконана на 82 сторінках та містить 36 рисунків, 0 таблиць, 35 джерел за переліком посилань, 1 додаток.

Мета роботи: є розробка програмного модуля комп’ютерно-інтегрованої системи реконволюції розмитих зображень.

Методи дослідження. Статистичні методи аналізу розподілу градієнтів зображень, методи цифрової фільтрації зображень на сонові фільтрів Вінера і Тихонова, ітераційні методи уточнення ядра розмиття на основі алгоритму Річардсона-Люсі.

Результати роботи та їх новизна. Запропонована вдосконалена методика реконволюції нечітких зображень. В результаті чого розроблено програмний модуль комп’ютерно-інтегрованої системи відновлення розмитих зображень на основі уточнення ядра розмиття методом Річардсона-Люсі з використанням попередньої фільтрації зображення за допомогою фільтра Тихонова.

Рекомендації по використанню результатів роботи: Програмний модуль на даному етапі може використовуватись для відновлення чіткості відзнятих фотографій та інших зображень. При цьому передбачено деякі початкові налаштування, які можуть впливати на якість кінцевого результату реконволюції.

Можливі напрямки розвитку. В подальшому програмний модуль може вдосконалюватись на сонові новітніх досліджень в галузі покращення якості зображень. Сам алгоритм може бути оптимізований для підвищення швидкодії. Крім того методика може бути вбудована в фотокамери для усунення розмиття фотографій в наслідків мікропереміщень фотокамери в руках користувача.

Ключові слова: ВІДНОВЛЕННЯ НЕЧІТКИХ ЗОБРАЖЕНЬ, ЯДРО РОЗМИТТЯ, АЛГОРИТМ РІЧАРДСОНА-ЛЮСІ, ЦИФРОВА ФІЛЬТРАЦІЯ ЗОБРАЖЕНЬ.

ABSTRACT

Work is executed on 82 pages and including 36 illustrations, 0 tables, 35 source after the list of references, 1 appendix.

Purpose of work: is the development of a software module for a computer-integrated system for the reconvolution of blurred images.

Research methods. Statistical methods for analyzing the distribution of gradients of images, methods of digital filtration of images based on filters Wiener and Tikhonov, iterative methods for clarifying the nucleus of blurring on the basis of the Richardson-Lucy algorithm.

Results of work and their novelty. An advanced technique for reconverting fuzzy images is proposed. As a result, a software module for the computer-integrated system for recovering blurred images was developed based on the clarification of the Richardson-Lucy blurry nucleus using a pre-filtering of the image using the Tikhonov filter.

Recommendations after the use of job performances: The program module at this stage can be used to restore the resolution of captured photographs and other images. It provides some initial settings that can affect the quality of the final result of the reconvolution.

Possible development. Subsequently, the software module can be improved on the basis of the latest research in the field of image quality improvement. The algorithm itself can be optimized to improve performance. In addition, the technique can be embedded in the camera to eliminate the blurring of the photos in the aftermath of the micro-displacement of the camera in the user's hands.

Keywords: BLIND IMAGE DECONVOLUTION, BLUR KERNEL, RICHARDSON-LUCY ALGORITHM, DIGITAL IMAGE FILTERING.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MAP – максимальне апостеріорне рішення

ШПФ – швидке перетворення Фур’є

API – Application Programming Interface (інтерфейс прикладного програмування).

GNU – вільна UNIX-подібна операційна система.

GUI – Graphical User Interface (графічний інтерейс користувача).

IDE – Integrated Development Environment (інтергроване середовище розробки).

JDK – Java Development Kit.

JRE – Java Runtime Environment.

JVM – Java Virtual Machine (віртуальна машина Java).

KDE – вільне графічне оточення графічного столу.

RL – алгоритм Річардсона -Люсі

PSF **–** Point Spread Function (функція розподілу точок).

SOA – Service-oriented architecture (сервісно-орієнтована архітектура).

SWT – Standard Widget Toolkit.

TIOBE – індекс для оцінки популярності мов програмування.

UML – Unified Modeling Language (уніфікована мова моделювання).

XML – Extensible Markup Language (розширювана мова розмітки).

ВСТУП

Актуальність теми. З інтенсивним розвитком інформаційних і комп'ютерних технологій швидкими темпами вдосконалюється фото і відеоапаратура та розширюються області їх застосування. Крім того, вдосконалення телекомунікаційних систем і комп'ютерних мереж дозволяє передавати великі об'єми інформації, що призвело до інтенсивного поширення відеоданих та різного роду зображень, в тому числі і фотознімків. В той же час паралельно йде швидкий розвиток теоретичних засад цифрової обробки зображень, що гармонійно доповнює цифрову апаратуру і дозволяє значно розширити її можливості та сфери застосування.

На сьогоднішній день фото та відеодані використовуються в промисловості, медицині, космосі, військовій сфері, охоронних системах, мистецтві та багатьох інших галузях. Формування чітких зображень, поліпшення їх якості та автоматизація різноманітних видів обробки зображень, включаючи зображення, що створюються електронними мікроскопами, рентгенівськими апаратами, томографами, штучними супутниками, безпілотними літальними апаратами, камерами відеоспостереження тощо, є предметом сучасних досліджень та розробок. При цьому гострою залишається проблема відновлення нечітких зображень, які мають різний характер та причини спотворень. В зв'язку з цим актуальними задачами є вдосконалення та створення нових методів цифрової обробки зображень, що включають в себе зокрема фільтрацію та деконволюцію.

Мета і завдання дослідження. Метою роботи є вдосконалення методів та алгоритмів деконволюції розмитих зображень та розроблення на їх основі програмного забезпечення комп'ютерно-інтегрованої системи.

Предмет дослідження є розмиті зображення, що являються одними з найбільш неприємних дефектів у фотографії,а також алгоритми деконволюції змазаних і розфокусованих зображень.

Об’єктом дослідження є методи та алгоритми відновлення чіткості розмитих зображень.

Методи дослідження базуються на методах фільтрації, методах ітераційного наближення розв’язків, теорії обробки сигналів і зображень.

Одержані результати та їх новизна. В роботі вдосконалено метод деконволюції нечітких зображень на базі визначення ядра розмиття методом Річардсона-Люсі. Розроблено програмний модуль комп’ютерно-інтегрованої системи реалізації запропонованого методу.

Напрямки подальшого розвитку. В подальшому планується оптимізувати метод деконволюції та алгоритми його реалізації за рахунок новітніх досягнень в галузі обробки зображень та здійснення попередньої обробки зображення для більш ефективного відновлення зображень

РОЗДІЛ 1
МАТЕМАТИЧНІ ОСНОВИ ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ

1.1. Модель процесу спотворення і відновлення зображення

Помилково вважається, що розмиття необоротна операція і інформація безповоротно втрачається, тому що кожен піксель перетворюється на пляму, все змішується, а при великому радіусі розмиття так і зовсім отримаємо однорідний колір по всьому зображенню. Це не зовсім так – вся інформація просто перерозподіляється по деякому закону і може бути однозначно відновлена з деякими застереженнями. Виняток становлять лише краї зображення шириною в радіус розмиття – там повноцінне відновлення неможливо.

Продемонструємо це, використовуючи невеликий приклад для одновимірного випадку – уявімо що у нас є ряд з пікселів зі значеннями: x_1 | x_2 | x_3 | x_4 ... – початкове зображення.

Після спотворення значення кожного пікселя підсумовується зі значенням лівого, тобто $x`_i = x_i + x_{i-1}$. Взагалі, треба ще поділити на 2, але опустимо це для простоти. В результаті маємо розмите зображення зі значеннями пікселів: $x_1 + x_0$ | $x_2 + x_1$ | $x_3 + x_2$ | $x_4 + x_3$ ... – розмите зображення. Тепер будемо пробувати відновлювати, віднімемо послідовно по ланцюжку значення за схемою – з другого пікселя перший, із третього результат другого, з четвертого результат третього і так далі, отримаємо: $x_1 + x_0$ | $x_2 - x_0$ | $x_3 + x_0$ | $x_4 - x_0$ ... – отримаємо відновлене зображення.

У підсумку замість розмитого зображення отримали вихідне зображення, до пікселів якого додана невідома константа x_0 зі знаком, що чергується. Це вже набагато краще – цю константу можна підібрати візуально, можна припустити, що вона приблизно дорівнює значенню x_1, можна автоматично підібрати з таким критерієм, щоб значення сусідніх пікселів “скакали” якомога менше і т.д. Але все міняється, як тільки додаємо

шум (який завжди є в реальних зображеннях). За описаною схемою на кожному кроці буде накопичуватися внесок шуму в загальну складову, що в підсумку може дати зовсім неприйнятний результат, але, як переконалися вище, відновлення цілком реальне навіть таким примітивним способом.

Кінцевою метою відновлення є підвищення якості зображення в деякому заздалегідь зумовленому сенсі. При відновленні робиться спроба реконструювати або відтворити зображення, яке було до цього спотворено, використовуючи апріорну інформацію про явище, яке викликало погіршення зображення. Тому методи відновлення засновані на моделюванні процесів спотворення і застосуванні зворотних процедур для відтворення вихідного зображення. Цей підхід зазвичай включає розробку критеріїв якості, які дають можливість об’єктивно оцінити отриманий результат.

Деякі методи відновлення зручно формулюються в просторовій області, в той час як для формулювання інших більше підходить частотна область. Наприклад, просторова обробка застосовна у випадку, коли єдиним джерелом спотворень є адитивний шум. З іншого боку, завдання відновлення змазаних зображень, важко піддається вирішенню в просторовій області з використанням масок малого розміру. У цьому випадку правильним підходом є використання частотних фільтрів, отриманих на основі різних критеріїв оптимальності; такі фільтри враховують також і наявність шуму.

Модель процесу спотворення передбачає дію деякого спотворюючого оператора H на вихідне зображення $f(x, y)$, що після додавання адитивного шуму дає спотворене зображення $g(x, y)$. Завдання відновлення полягає у побудові деякого наближення $f'(x, y)$ вихідного зображення по заданому (спотвореному) зображенню $g(x, y)$, з деякою інформацією щодо спотворюючий оператор H, і деякою інформацією щодо адитивного шуму $n(x, y)$. Необхідно, щоб наближення було якомога ближче до вихідного зображення, і, в принципі, чим більше інформації відомо про оператор H і функцію n, тим ближчою буде функція $f'(x, y)$ до функції $f(x, y)$. В основі

підходу лежить використання операторів (фільтрів), що відновлюють зображення.

Спотворене зображення може бути представлене в просторовій області у вигляді:

$$g(x, y) = h(x, y) * f(x, y) + n(x, y), \quad (1.1)$$

де $h(x, y)$ – функція, що представляє спотворюючий оператор в просторовій області, а символ “*” використовується для позначення згортки. Відомо, що згортка в просторовій області еквівалентна множенню в частотній області, тому рівність (1.1) може бути записана в частотній області:

$$G(u, v) = H(u, v)F(u,v) + N(u,v), \quad (1.2)$$

де великими літерами позначені функції – Фур’є-образи відповідних функцій в (1.1).

Лінійні трансляційно-інваріантні моделі можуть бути використані для наближеного опису багатьох типів спотворень. Хоча нелінійні і трансляційно-неінваріантні методи є більш загальними (і зазвичай більш точними), але їх використання часто приводить до невирішуваних або дуже важко вирішуваних чисельними методами проблем. Оскільки спотворення являє собою результат згортки, то для відновлення необхідно знайти такий фільтр, застосування якого призводило б до зворотного процесу. Тому для позначення лінійного процесу відновлення часто використовується термін реконструкція (деконволюція) зображень. Аналогічно, фільтри, що використовуються для відновлення часто називаються реконструюючими фільтрами.

Існує три основні способи оцінки спотворюючої функції (ядра спотворюючого оператора) для подальшого її використання при відновленні зображень: візуальний аналіз, експеримент і математичне моделювання.

Внаслідок того, що дійсна спотворююча функція нечасто буває відома повністю, процес відновлення зображення з використанням наближення даної функції, отриманого деяким чином, іноді називають реконструкцією “наосліп”.

Припустимо, що є спотворене зображення, але інформація про спотворюючу функцію H відсутня. Один із способів оцінити цю функцію полягає у виділенні інформації безпосередньо з зображення. Наприклад, якщо зображення є розмитим, можна розглянути його невеликий фрагмент, що містить просту структуру, таку як частина деякого об’єкту і фон. Для того щоб зменшити вплив шуму спостереження, слід вибрати ту область зображення, яка містить корисний сигнал великої амплітуди. Використовуючи яскравості об’єкта і фону, можна приблизно побудувати нерозмиті зображення тих же розмірів і з тими ж особливостями, що і розглянута частина вихідного зображення. Позначимо дану частину зображення як $g_s(x, y)$ і відновлене зображення (яке насправді являє собою наше наближення для частини неспотвореного зображення в розглядуваній області) як $\hat{f}_s(x, y)$. Далі, припускаючи, що вплив шуму порівнянно малий в силу нашого вибору області з великим корисним сигналом маємо:

$$H_s(u, v) = \frac{G_s(u, v)}{\hat{F}_s(u, v)} \quad (1.3)$$

Виходячи з властивостей функції $H_s(u, v)$, можна зробити висновки про властивості повної спотворююючої функції $H(u,v)$, якщо вважати, що спотворення передбачаються трансляційно-інваріантні.

1.2 Методи фільтрації розмитих зображень

Інверсна фільтрація. У цьому пункті зробимо перший крок у вирішенні задачі відновлення зображень, спотворених оператором H, ядро

(спотворююча функція) якого задано або визначено за допомогою методів, розглянутих в попередньому підпункті. Найпростішим способом відновлення є інверсна фільтрація, яка передбачає отримання оцінки $\hat{f}_s(u,v)$ Фур'є-перетворення вихідного зображення поділом Фур'є-перетворення спотвореного зображення на частотне представлення спотворюючої функції:

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)} \quad (1.4)$$

Ділення в (1.4) розуміється як поелементне. Підставивши в (1.4) вираз для $G(u, v)$, отримаємо:

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)} \quad (1.5)$$

З цього видно, що навіть знаючи спотворюючу функцію, неможливо точно відновити неспотворене зображення (зворотне Фур'є-перетворення функції $F(u, v)$), оскільки функція $N(u, v)$ є перетворенням Фур'є випадкової величини і невідома. Є ще одна проблема. Якщо функція $H(u, v)$ приймає нульові або близькі до нульових значення, то вклад другого доданка в правій частині (1.5) може стати домінуючим. І на практиці ця ситуація переважає.

Один із способів обійти зазначену проблему полягає в тому, щоб обмежити частоти фільтру значеннями поблизу початку координат. Як відомо, значення $H(0,0)$ дорівнює средньому значенню функції $h(x, y)$ і зазвичай є найбільшим значенням $H(u, v)$ в частотній області. Тому обмежуючи розгляд частот поблизу початку координат, зменшуємо імовірність зустріти нульове значення.

Даний метод має дуже обмежені можливості і придатний для використання лише на малозашумлених зображеннях, але він є основою для побудови фільтрів кращої якості.

Вінерівська фільтрація. Розглянутий вище метод інверсної фільтрації не забезпечує коректної роботи по відношенню до шуму. У цій частині розглянемо метод, який поєднує в собі врахування властивостей спотворюючої функції і статистичних властивостей шуму в процесі відновлення. Метод заснований на розгляді зображень і шуму як випадкових процесів, і завдання ставиться таким чином: знайти таку оцінку $\hat{f}$ для неспотвореного зображення f, щоб середньоквадратичне відхилення цих величин одна від одної було мінімальним. Середньоквадратичне відхилення e задається формулою:

$$e^2 = E\{(f - \hat{f})^2\}, \tag{1.6}$$

де E{·} позначає математичне сподівання свого аргумента.

Припускається, що виконуються наступні умови:

- шум і вхідне зображення не корельовані між собою;
- або шум або неспотворене зображення мають нульове середнє значення;
- оцінка лінійно залежить від спотвореного зображення.

При виконанні цих умов мінімум середньоквадратичного відхилення (1.6) досягається на функції, яка задається в частотній області виразом:

$$\hat{F}(u,v) = \left(\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + S_n(u,v)/S_f(u,v)}\right)G(u,v) \tag{1.7}$$

де $H(u,v)$ – спотворююча функція (її частотне подання);

$H^*(u,v)$ – комплексне спряження H(u, v);

$|H(u,v)|^2 = H^*(u, v)H(u, v)$;

$S_n(u,v) = |N(u, v)|^2$ – енергетичний спектр шуму;

$S_f(u,v)=|F(u,v)|^2$ – енергетичний спектр неспотвореного зображення.

$G(u, v)$ – Фур’є-перетворення спотвореного зображення

Причому остання рівність вірна внаслідок того, що добуток комплексного числа на комплексно-спряжене рівне квадрату модуля. Наведений результат був отриманий Н. Вінером [1], і метод відомий як оптимальна фільтрація за Вінером. Фільтр, представлений виразом всередині дужок, часто називають фільтром мінімального середньоквадратичного відхилення або вінеровським фільтром.

Відновлене зображення просторової області отримується шляхом застосуванням зворотного перетворення Фур'є до оцінки $\hat{F}(u, v)$. Відзначимо, що якщо шум дорівнює нулю, то його енергетичний спектр згортається в нуль, і вінерівська фільтрація в цьому випадку зводиться до інверсної фільтрації.

Коли маємо справу з білим шумом, спектр якого $|N(u, v)|^2$ є постійною функцією, відбуваються відповідні спрощення. Однак, спектр неспотвореного зображення рідко буває відомий. У тих випадках, коли спектри шуму і неспотвореного зображення невідомі і не можуть бути оцінені, часто використовується підхід, що являється апроксимацією виразу (1.7) виразом:

$$\hat{F}(u,v) = \left(\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right) G(u,v) \qquad (1.8)$$

де К – певна константа.

Для відновленння зображень в роботі буде використовуватись саме цей вираз.

Фільтр autolevels. Еквалізацію (лінеаризацію) гістограми проводять в тому випадку, коли в зображенні є багато пікселів зі схожими яскравостями, і мало пікселів з іншими яскравостями. На гістограмі видно, що на деяких проміжках яскравостей згруповано багато пікселів, в той час як деякі

проміжки яскравостей майже не зайняті. При цьому деталі зображення, які зображені цими кольорами, складно розрізнити. Натомість існують такі проміжки яскравості, пікселів з якими взагалі немає на зображенні. Ці вільні проміжки яскравості можна «зайняти» для покращення якості зображення. Для цього роблять еквалізацію гістограм.

Якщо маємо піксель початкового зображення з яскравістю b_k, яка є k-им рівнем яскравості на гістограмі (k=0…N-1), то яскравість відповідного пікселя результуючого зображення буде розраховуватися:

$$r_k = \sum_{p=0}^{k} H(b_p) = \sum_{p=0}^{k} \frac{N_p}{N} \quad (1.9)$$

В результаті еквалізації гістограми яскравості пікселів на ній будуть розподілені рівномірно по всій шкалі яскравостей.

На рисунку 1.1 наведене зображення, яке виглядає дуже темним.

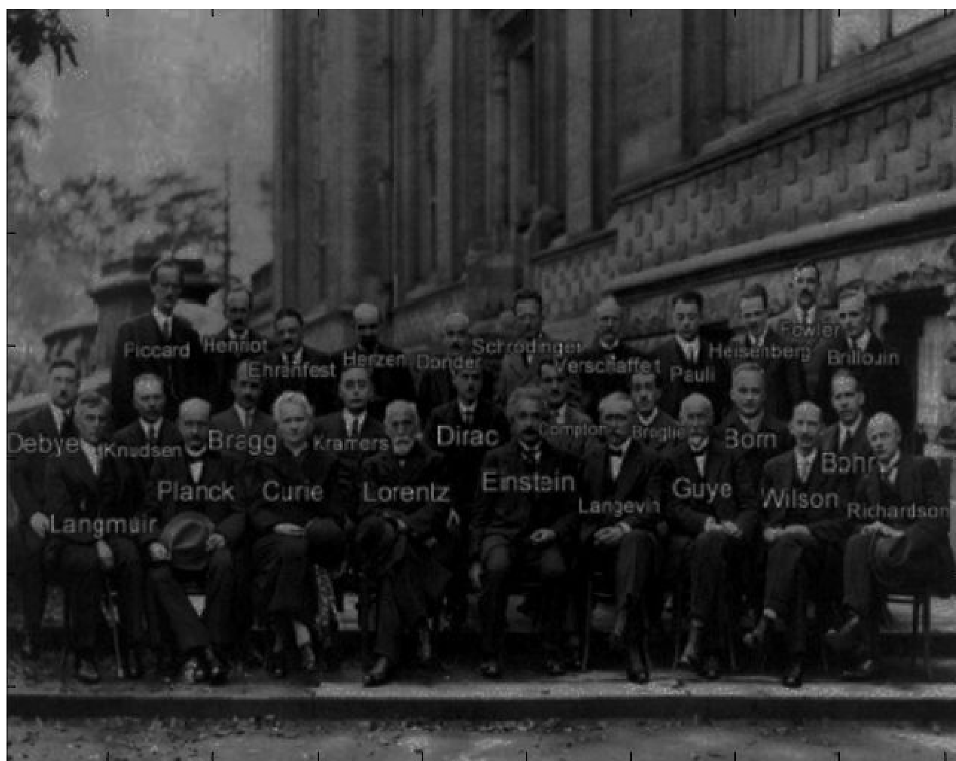


Рисунок 1.1 – Темне зображення

Дрібні деталі предметів та людей на ньому розрізнити складно, оскільки вони зображені схожими темними кольорами, які мало відрізняються один від одного. Гістограма цього зображення наведена на рисунку 1.2.

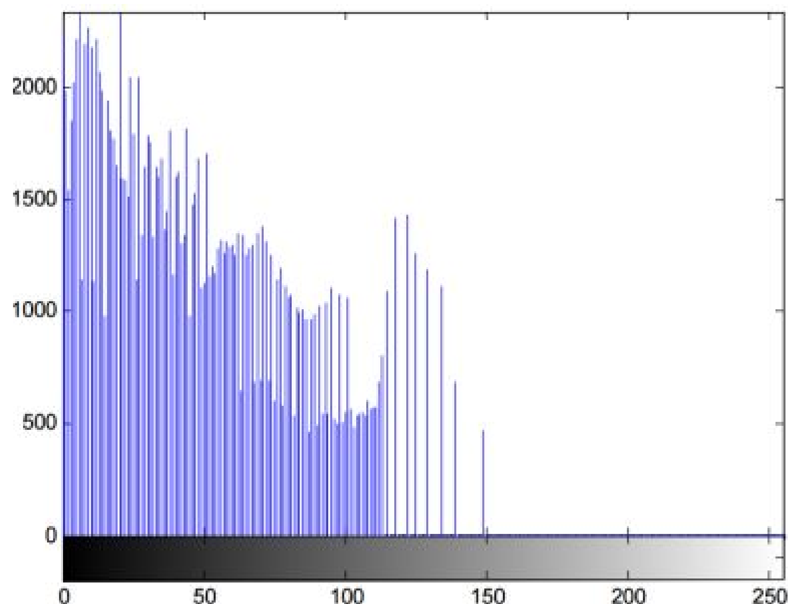


Рисунок 1.2 – Гістограма темного зображення

На ній видно, що багато пікселів знаходяться в лівій частині шкали кольорів, що відповідає темним кольорам. Водночас, права частина шкали майже не зайнята, тобто світлих пікселів на зображенні немає. Цей вільний проміжок гістограми можна використати, щоб перенести туди яскравості деяких пікселів. Якщо гістограму цього зображення «розтягнути» на весь доступний діапазон яскравостей, то пікселі, які раніше мали дуже схожі кольори (їх яскравості знаходились близько на шкалі яскравостей), будуть віддалені один від одного на більшу яскравості.

Якщо подивитись на зображення, видно, що діапазон яскравостей пікселів, які присутні на зображенні, розширився: на зображенні тепер є і темні, і світлі пікселі (рисунок 1.3).

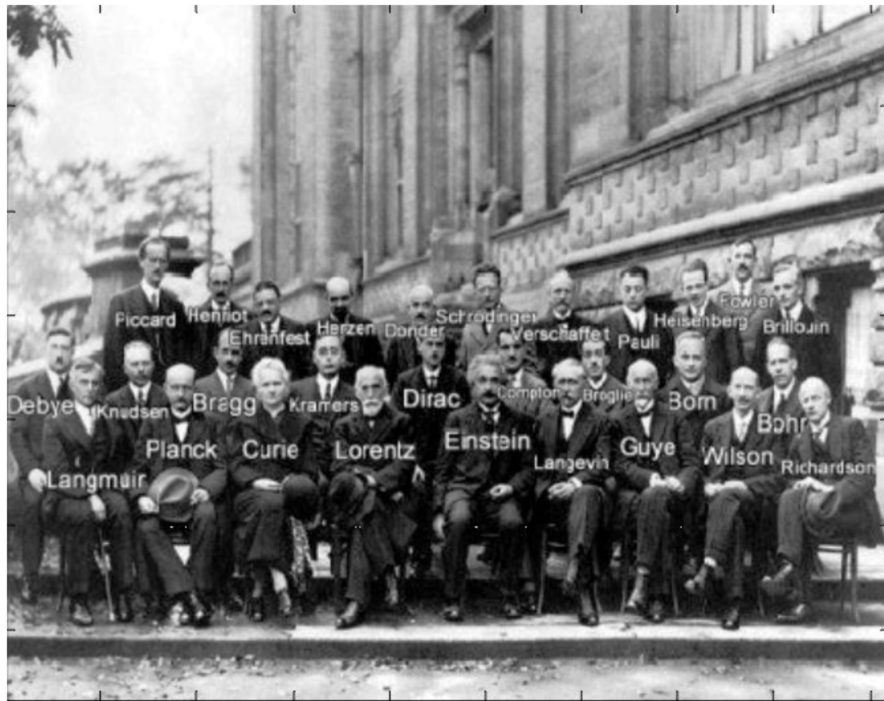


Рисунок 1.3 – Темне зображення після еквалізації

Тепер стало легше розрізнити деталі зображення, оскільки вони зображені більш контрастно. На гістограмі (рисунок 1.4) видно, що на зображенні присутні пікселі всіх яскравостей, і весь діапазон яскравостей тепер зайнятий.

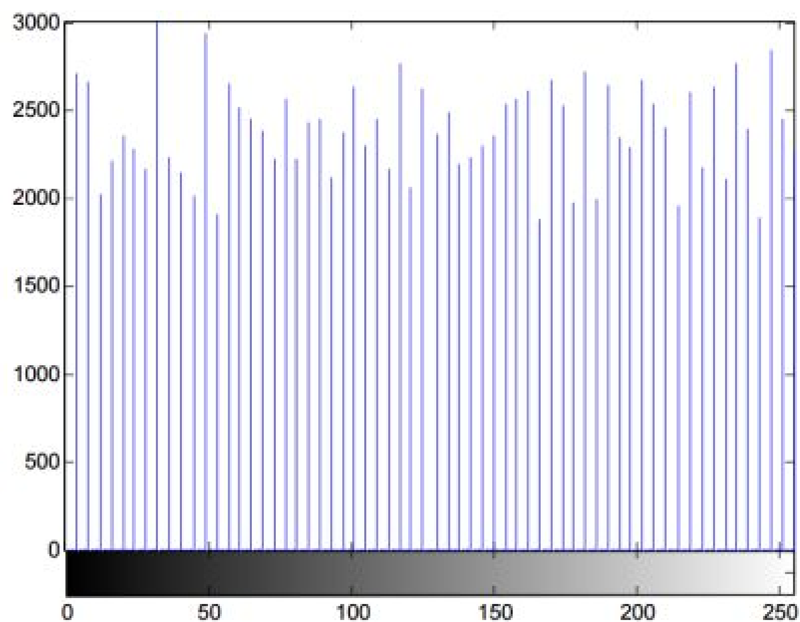


Рисунок 1.4 – Гістограма еквалізованого зображення

Перевагою еквалізації гістограм є те, що цей метод легко автоматизується і не вимагає задавання ніяких додаткових параметрів для отримання покращеного зображення. Розрахунки для еквалізації гістограм є достатньо нескладні.

1.3. Функції розподілу точок для відновлення чіткості зображень

Способи отримання функції розподілу точок PSF (Point Spread Function):

Візьмемо за основу описаний вище фільтр Вінера – взагалі кажучи, існує безліч інших підходів, але всі вони дають приблизно однакові результати. Так що все описане нижче буде справедливо і для інших методів деконволюції.

Основне завдання – отримати оцінку функції розподілу точки (PSF). Це можна зробити декількома способами.

Моделювання є дуже непростими і трудомістким, тому що сучасні об'єктиви складаються з більше ніж десятка різних лінз і оптичних елементів, частина з яких має асферичну форму, кожен сорт скла має свої унікальні характеристики заломлення променів з тією чи іншою довжиною хвилі. У підсумку завдання коректного розрахунку поширення світла в такій складній оптичній системі з урахуванням впливу діафрагми, перевідбиттів і стає практично неможливим. І вирішення його, мабуть, доступне тільки розробникам сучасних об'єктивів.

Безпосереднє спостереження. Оскільки PSF – це те, у що перетворюється кожна точка зображення. Тобто якщо сформуємо чорний фон і одну білу точку на ньому, а потім сфотографуємо це з потрібним значенням розфокусування, то отримаємо безпосередньо вид PSF. Здається просто, але є багато нюансів і тонкощів.

Обчислення або непряме спостереження. З формули (1.2) розв'язок

видно зразу – потрібно мати вихідне *F(u, v)* і спотворене *G(u, v)* зображення. Тоді поділивши Фур'є-образ спотвореного зображення на Фур'є-образ вихідного зображення отримаємо шукану PSF.

Ідеальний об'єктив має PSF у вигляді круга, відповідно кожна точка перетворюється в круг деякого діаметру. До речі, це для багатьох несподіванка, тому з першого погляду здається, що дефокус просто розмазує все зображення. Це ж пояснює і те, чому фотошопне розмиття Гаусса зовсім не схоже на той малюнок фону (його ще називають боке), який бачимо у об'єктивів. Насправді це два різних типи розмиття – по Гаусу кожна точка перетворюється на нечітку пляму (дзвін Гаусса), а дефокус кожну точку перетворює на круг. Відповідно і різні результати.

Але ідеальних об'єктивів неіснує і в реальності отримуємо те чи інше відхилення від ідеального кола. Саме це і формує неповторний малюнок боке кожного об'єктива, змушуючи фотографів витрачати багато грошей на об'єктиви з гарним боке. Боке можна умовно розділити на три типи (рисунок 1.5):

- нейтральне – це максимальне наближення до кола;
- м'яке – краї мають меншу яскравість, ніж центр;
- жорстке – краю мають велику яскравість, ніж центр.

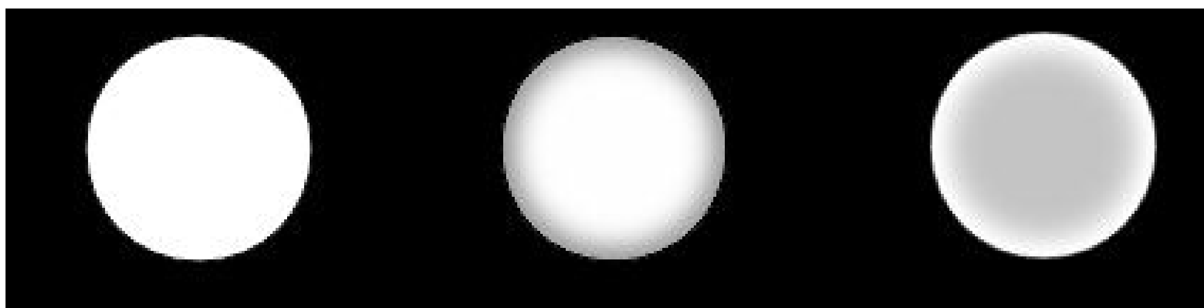


Рисунок 1.5 – Різні типи боке.

Більше того, тип боке – м'яке або жорстке залежить ще й від того, передній це фокус чи задній. Тільки нейтральне боке не змінюється від виду фокусу.

Але – оскільки кожному об’єктиву притаманні ті чи інші геометричні спотворення, то вид PSF залежить ще й від положення. У центрі зображення – круг, по краях – еліпси та інші сплюснуті фігури. Це добре видно на рисунку 1.6.

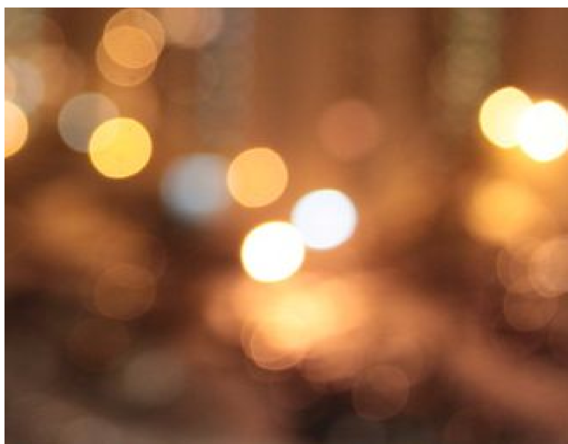


Рисунок 1.6 – Залежність PSF від положення об’єктива

Розглянемо докладніше метод непрямого спостереження отримання PSF. Для цього, як писалося вище, потрібно мати вихідне $F(u, v)$ і спотворене $G(u, v)$ зображення. Отримати їх дуже просто – необхідно поставити фотоапарат на штатив і зробити один різкий і один розмитий знімок одного і того зображення. Далі за допомогою ділення Фур’є-образу спотвореного зображення на Фур’є-образ вихідного зображення отримаємо Фур’є-образ шуканої PSF.

Після чого застосувавши зворотне перетворення Фур’є отримаємо PSF в прямому вигляді. В результаті отримано PSF, зображене на рисунку 1.7.

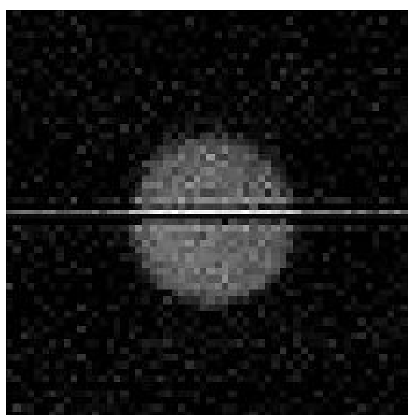


Рисунок 1.7 – Отримане PSF методом ділення в частотній області.

Результат посередній – дуже багато шумів і деталі PSF видно не так добре. Тим не менш, метод має право на існування.

Описані методи можна і потрібно використовувати для побудови PSF при відновленні розмитих зображень. Оскільки від того, наскільки ця функція наближена до реальної безпосередньо залежить якість відновлення початкового зображення. При розбіжності передбачуваної та реальної PSF будуть спостерігатися численні артефакти у вигляді "дзвону", ореолів і зниження чіткості. У більшості випадків передбачається форма PSF у вигляді кола, проте для досягнення максимального ступеня відновлення рекомендується попрацювати з формою цієї функції, спробувавши кілька варіантів від поширених об'єктивів – як було видно, форма PSF може варіюватися в значній мірі залежно від діафрагми, об'єктива і інших умов.

Ще одна проблема полягає в тому, що якщо безпосередньо застосувати фільтр Вінера, то на краях зображення буде своєрідний "дзвін". Його причина, полягає в наступному – коли робиться деконволюція для тих точок, які розташовані на краях, то при сумуванні не вистачає пікселів, які знаходяться за краями зображення і вони приймаються або рівним нулю, або беруться з протилежного боку (залежить від реалізації фільтра Вінера і перетворення Фурє). Вигляд цього явища продемонстровано на рисунку 1.8.

Рисунок 1.8 – "Дзвони" на зображенні в результаті деконволюції

Одне з рішень, щоб уникнути цього полягає в доробці країв зображення. Вони розмиваються за допомогою тієї ж самої PSF. На практиці це реалізується наступному чином – береться вхідне зображення *F(x, y)*, розмивається за допомогою PSF і виходить *F’(x, y)*, потім підсумкове вхідне зображення *F’’(x, y)* формується додаванням *F(x, y)* і *F’(x, y)* з використанням вагової функції, яка на краях приймає значення 1 (точка цілком береться з розмитого *F’ (x, y)*), а на відстані рівному (або більшому) радіусу PSF від краю зображення приймає значення 0.

2. ПОКРАЩЕНА МЕТОДИКА ВІДНОВЛЕННЯ РОЗМИТИХ ЗОБРАЖЕНЬ

2.1 Побудова моделі відновлення розмитого зображення

В роботі представляє покращену методику відновлення зображення завдяки видалення ефектів тремтіння дзеркальних камер. Це здійснено за рахунок використання недавніх дослідження в статистиці зображень, згідно яких фотографії природних сцен зазвичай підпорядковуються дуже специфічним розподілам градієнтів зображення. По-друге, в роботі використано результати праць Мискина та МакКей [2], запропонувавши байєсівський підхід, який враховує невизначеність у невідомих, дозволяючи знайти ядро розмивання, що передбачається розподілом імовірних зображень. З врахуванням характеристик цього ядра, зображення потім реконструюється за допомогою стандартного алгоритму деконволюції, хоча вважається, що для цієї фази реконструкції існує можливість істотного поліпшення. Припустимо, що все розмиття зображення можна описати як єдину згортку; тобто відсутній значний паралакс, обертання камери зображення на площині зображення невелика, і жодна частина сцени не рухається одна відносно одної під час експозиції. Запропонований підхід в даний час вимагає невеликої кількості налаштувань користувача. Отримані реконструкції містять артефакти, особливо коли зазначені припущення порушуються; однак вони можуть бути прийнятними для споживачів у деяких випадках, і професійний дизайнер може виправити отримані результати. Тоді як, оригінальні зображення, як правило, непридатні для використання, без використання подібних методів які розглядаються в роботі, що можуть значно покращити якість знімків, які в іншому випадку були б повністю втрачені.

Коли необхідно відновите розмите зображення, якщо ядро розмивання невідоме, тоді проблема вважається "сліпою". Ці питання досліджували, Кундюр і Хатзінакос [3]. Існуючі методи «сліпої» деконволюції зазвичай передбачають, що ядро розмивання має просту параметричну форму, таку як гаусові або низькочастотні Фур'є-компоненти. Проте, як показують наведені приклади, ядра розмивання, індуковані під час руху камери, мають не прості форми і часто не тільки мають прямолінійний характер як показано в п.1. 3, а є кривими, які дуже часто містять гострі кути. Подібні спрощені припущення, як правило, виконуються для вхідного зображення, наприклад, застосовуючи квадратну регуляризацію. Такі припущення можуть запобігти виникненню високих частот (наприклад, країв) у процесі реконструкції.

Деякі методи запропоновані Джалобеуном 2002[4] і Нееламані [5], що використовують строгі закони розмиття зображень з обмеженнями вейвлет-домену, не працюють для складних ядер розмивання в складніших випадках наведених в прикладах. Методи деконволюції були розроблені для астрономічних зображень Галом, Річардсон, Цумурая і Заровіном [6-9], що мають статистичні дані, які цілком відрізняються від природних сцен, які розглядаються в цій роботі.

Виконання «сліпої» деконволюції в цій області, як правило, є простою, оскільки розмитий образ ізольованої зірки виявляє функцію розповсюдження по точкам. Інший підхід полягає в припущенні, що існує декілька доступних зображень однієї сцени запропонований Баскл, Рав-Ача і Пелег [10-11]. До апаратних підходів відносяться: оптично стабілізовані лінзи [12], спеціально розроблені датчики CMOS Ліу і Гамала [13], а також гібридні системи зображень Бен-Езра і Наяра [14]. Для випадку камер без додаткового обладнання, необхідно застосовувати алгоритмічні методи покращення якості зображень на зображеннях. Останні досягнення з комп'ютерного бачення показала корисність відновлення розмитих природних зображень у різноманітних областях застосування, включаючи розпізнавання Рота і Блека [15], суперрезоляцію Таппена [16], внутрішні зображення Вейса [17],

відеоматеріали Апостолофа та Фіцгіббона, [18], втілення Левіна. [19] і відокремлюючи віддзеркалення Левіна і Вейса [20]. Кожний з цих методів є фактично "не сліпим", оскільки процес формування зображення (наприклад, ядро розмивання при суперрезолізації) вважається відомим заздалегідь. Міскін і МакКей [2] виконують сліпе деконволюцію на зображеннях лінійного образу, використовуючи попередню інтенсивність синього пікселя. Результати показані для зображень з не дуже сильного синтезованим розмиттям. В роботі застосовано аналогічну варіаційну схему для природних зображень, використовуючи градієнти зображення замість інтенсивностей, і покращено алгоритм, щоб досягти результатів для фотографічних зображень з значним розмиттям.

Алгоритм приймає в якості входу розмитий вхідний образ B, який вважається сформованим шляхом згортки ядра розмивання K з латентним зображенням L плюс шум N:

$$B = K \otimes L + N \quad (2.1)$$

де $\otimes$ – дискретна згортка (з неперіодичними граничними умовами),

N – шум датчика на кожному пікселі.

Припустимо, що піксельні значення зображення лінійно пов'язані з випромінюванням сенсора. Латентне зображення L являє собою зображення, яке було б отримане, якби фотокамера залишилася цілком нерухомою. Задача полягає в тому, щоб відновити зображення L маючи розмите зображення B при невідомому специфічному значенні розмиття K.

Для того, щоб оцінити приховане зображення з таких обмежених вимірювань, важливо мати певне уявлення про те, які зображення є апріорі вірогіднішими.

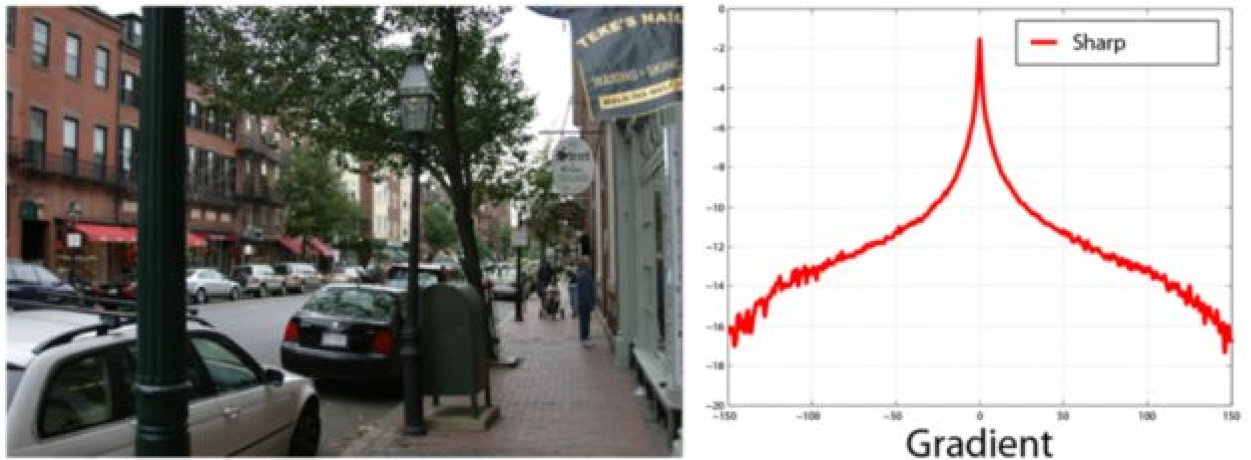
Дослідження показали, що незважаючи на те, що реальні зображення мають великий діапазон значень окремих пікселів, градієнти цих значень мають вигляд розподілу з повільно зменшуваними межами (Heavy-tailed distribution) [21]. Такий розподіл має пік в околиці нуля (рисунок1)і, на відміну від гаусового розподілу, має значно більші імовірності великих значень. Це збігається з інтуїтивним уявленням, що на реальних зображеннях в більшості випадків присутні великі області більш-менш постійної яскравості, які закінчуються об'єктами з різкими і середніми перепадами яскравості.

На рисунку 2.2 показано приклад природного зображення та гістограми його градієнтних величин. Розподіл показує, що зображення містить переважно малі чи нульові градієнти, але кілька градієнтів мають великі значення.

Розроблені за останній час способи обробки зображень на основі важкоатлетичних розподілів дають найсучасніші результати при зніманні зображення Рош, Блек і Сімонцеллі [15, 22] з великою роздільною здатністю Таппен [15.].

В той час як методи, засновані на використанні розподілу Гауса (включаючи методи, що використовують квадратичні оцінки), реконструюють зображення з надто плавними переходами. На рисунку 2.1 представлено розподіл градієнтів кольорів з переважним попаданням їх в околі нуля. Такий розподіл було обрано, оскільки він забезпечує краще наближення до емпіричного розподілу, одночасно дозволяючи використати оцінку розмиття для запропонованого алгоритму.

В реалізації алгоритму можна виділити два основних кроки. По-перше, ядро розмиття оцінюється з вхідного зображення. Процес оцінки виконується грубо, щоб уникнути локальних мінімумів. По-друге, використовуючи отримане орієнтовне ядро, застосовується стандартний алгоритм деконволюції для оцінки латентного (нерозпізнаного) зображення.



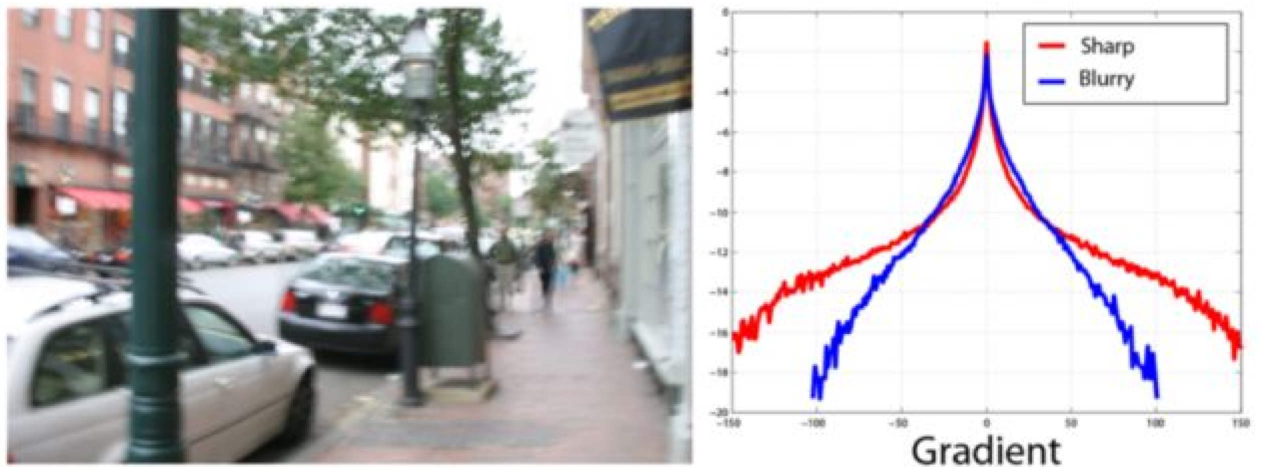


Рисунок 2.1 – Зліва представлена натуральна сцена. Справа червоним кольором показаний розподіл градієнта кольорів всередині сцени. Вісь *y* має логарифмічний масштаб для демонстрації довгих хвостів розподілу. Результат послідовного наближення Гаусса для розподілу градієнта показаний синім кольором.

Користувач задає чотири вхідні величини до алгоритму: розмитий малюнок B, початковий фрейм у розмитому зображенні, верхня межа розміру ядра розмиття (у пікселях) і початкове припущення щодо орієнтації ядра розмиття (горизонтальне або вертикально). Детальні рекомендації про те, як вказати ці параметри, наведено в пункті 2.2.

Крім того, потрібно, щоб вхідний образ B був перетворений у лінійний колірний простір перед обробкою. В представлених експериментах застосована зворотна гамма-корекція з $\gamma = 2{,}2$. Для того, щоб оцінити очікуване ядро розмиття, об'єднано всі канали кольорів оригінального зображення в користувальницькому фреймі, щоб отримати розмитий фрейм у градаціях сірого P.

2.2 Оцінка ядра розмиття

Згідно заданого розмитого фрейму P у градаціях сірого, оцінюється ядро розмивання K і приховане зображення фрейма L_p, знаходячи значення з найбільшою ймовірністю, керуючись попередньою статистикою L. Оскільки ці статистичні дані базуються на градієнтах зображення, а не на інтенсивності, оптимізація здійснюється в градієнтному просторі, використовуючи ∇L_p і ∇P, градієнти L_p і P. Оскільки згортка є лінійною операцією, градієнти фрейма ∇P повинні бути рівними згортці латентних градієнтів і ядра: $\nabla P = \nabla Lp \otimes K$, плюс шум.

Припустимо, що цей шум є гаусівським з дисперсією σ^2. Як вказувалось вище, попереднє значення $p(\nabla Lp)$ на градієнтах латентного зображення являє собою поєднання C нульових значень Гаусівського розподілу в околі нуля (з дисперсією v_c та вагою для c-ої Гаусівської плями). Використовується розрідження попереднього значення $p(K)$ для ядра, який заохочує нульові значення в ядрі, і вимагає, щоб всі записи були позитивними. Зокрема, попереднє значення для ядра – це поєднання D експонентних розподілів (з коефіцієнтами масштабу λ_d і вагами π_d для d-го

компонента). Враховуючи вимірювані градієнти зображення ∇P, можна записати наступний розподіл над невідомими за правилом Байєса:

$$p(K,\nabla L_p \big|\nabla P) \quad \infty \quad p(\nabla P\big|K, \nabla L_p)p(K))$$

$$= \prod_i N(\nabla P(i)\big|(K \otimes \nabla L_p(i)),\sigma^2) \qquad (2.2)$$

$$\prod_i \sum_{c=1}^{C} \pi_c N(\nabla L_p(i)\big|0,v_c)\prod_j \sum_{d=1}^{D} \pi_d E(K_j \big|\lambda_d , \qquad (2.3)$$

де i – індекс пікселі зображення;

j – індекси елементів ядра розмиття.

N і E – Гаусівський та експоненціальний розподіли відповідно.

Для прискорення вважається, що градієнти в ∇P незалежні один від одного, а також елементи в ∇L_p і K.

Прямий підхід до деконволюції – це визначення максимально апостеріорного рішення (MAP), яке знаходить ядро K і градієнти прихованого зображення ∇L, що максимізує $p(K, \nabla Lp/\nabla P)$. Це еквівалентно вирішенню задачі наближення методом найменших квадратів, який дозволяє здійснити приближення до даних і мінімізувати малі градієнти. Спроба використати суміщений градієнтний пошук показала, що алгоритм виявився не вдалим. Така інтерпретація полягає в тому, що цільова функція MAP намагається мінімізувати всі градієнти (навіть великі), тоді як природні зображення містять великі градієнти. Отже, алгоритм дає двотоновий образ, оскільки практично всі градієнти дорівнюють нулю. Якщо зменшити

дисперсію шумів (тим самим збільшуючи вагу значень, що відповідають даним), то алгоритм дає дельта-функцію для *K*, який точно відповідає розмитому зображенню, але без будь-якого покращення розмиття. Крім того, знайдена цільова функція MAP є дуже чутливою до небажаних місцевих мінімумів.

Проте, такий підхід дозволяє апроксимувати повний постеріорний розподіл $p(K, \nabla Lp|\nabla P)$, а потім обчислити ядро *K* з максимальною граничною ймовірністю. Цей спосіб вибирає ядро, яке є найбільш імовірним щодо розподілу можливих знімків прихованого зображення, таким чином уникаючи перебільшення, яке може виникнути при виборі єдиної "найкращої" оцінки зображення.

Щоб ефективно обчислити цю апроксимацію, приймається варіаційний байєсівський підхід Джордана [23], який обчислює розподіл $q(K, \nabla L_p)$, який апроксимує попередній $p(K, \nabla L_p|\nabla P)$. Зокрема, такий підхід грунтується на алгоритмі Міскіна та МакКея [2] для «сліпої» деконволюції зображень. Використовується факторизоване представлення: $q(K, \nabla L_p)=q(K)q(\nabla L_p)$. Для градієнтів прихованого зображення це наближення є Гаусівського щільністю, тоді як для невід'ємних елементів ядра розмиття це – виправлена Гаусівська пляма.. Розподіл для кожного початкового градієнта та елемента ядра розмиття представляється масивом середнього значення та дисперсії.

Згідно Міськіна та МакКея [2] також розглядалася дисперсію шуму σ^2 як невідоме протягом процесу оцінювання, тим самим звільняючи користувача від налаштування цього параметра. Це дозволяє значення шумів змінювати під час оцінки: на ранньому етапі процесу обмеження доступу до даних є більш вільним і стає все жорсткішим в подальшому, а знайдені рішення стають все кращими. Попередньо приймемо σ^2 у вигляді гамма-розподілу оберненої дисперсії, що має гіпер-параметри a, b:

$p(\sigma^2|a,b) = \Gamma(\sigma^{-2}|a,b)$. Змінна задня частина σ^2 є іншим гамма-розподілом $q(\sigma^{-2})$. Варіаційний алгоритм мінімізує цільову функцію, що відображає відстань між апроксимаційним та істинним розподілом, у відповідності до міри:

$$KL(q(K,\nabla L_p,\sigma^{-2})\|p(K,\nabla L_p|\nabla P). \qquad (2.4)$$

Отримані незалежні значення елементів дозволяють визначити цільову функцію C_{KL}:

$$< log\left[\frac{q(\nabla L_p)}{p(\nabla L_p)}\right] > q(\nabla L_p) + < log\left[\frac{q(K)}{p(K)}\right] > q(K) + < log\left[\frac{q(\sigma^{-2})}{p(\sigma^2)}\right] > q(\sigma^{-2}), \qquad (2.5)$$

де $<>q(0)$ Означає очікування по відношенню до $q(0)^2$.

Оскільки ∇P кратне, то воно опускається з рівняння. Тоді цільова функція мінімізується наступним чином. Значення розподілів $q(K)$ і $q(\nabla L_p)$ беруться для ініціалізації значень K і ∇L_p, при цьому дисперсія приймає великі значення, що не дає впевненості в початковій оцінці.

Потім параметри розподілу оновлюються по черзі в напрямку зменшення координат. Один з них змінюється незначно, в той час як інший визначається з попередньої моделі. Оновлення виконуються шляхом обчислення оптимального значення параметрів закритого формату та пошуком рядка у напрямку цих оновлених значень. Оновлення повторюються доти, поки зміна C_{KL} стане незначною. Значення розподілу $q(K)$ потім приймається як кінцева величина для K. Така реалізація адаптує вихідний код, запропонований Міскіном та МакКеєм [2].

У викладеному вище формулюванні знехтувано можливістю присутності насичених пікселів у зображенні та незручною нелінійністю, що вносить в представлену модель деякі похибки. Оскільки їх врахування

сильно ускладнює модель, перевага віддається просто маскуванню насичених областей зображення під час процедури виводу.

Для змінного фрейму, компоненти $C=D=4$ були використані в основі K і ∇L_p. Параметри градієнтів попереднього латентного зображення π_c, v_c оцінювали за допомогою одного зображення вулиці, показаного на рисунку 2.1, з використанням EM.

Оскільки статистика зображення змінюється в масштабі, кожен рівень шкали має свій набір попередніх параметрів. Це було використано для всіх експериментів. Параметри для попередніх елементів ядра розмиття оцінювалися за допомогою невеликого набору малозашумлених ядер, взятих з реальних зображень.

2.3 Реалізація запропонованого алгоритму та аналіз результатів його роботи

Алгоритм, описаний у попередньому пункті, залежить від локальних мінімумів, особливо для великих ядер розмиття. Оцінка виконується шляхом зміни роздільної здатності в грубому і точному вигляді. Самий грубий рівень – ядро K розмірністю 3×3. Щоб забезпечити правильний початок алгоритму, ми вручну вказуємо початкове ядро розмиття 3×3 на один із двох простих візерунків описаних нижче.

Початкова оцінка для зображення латентного градієнта здійснюється шляхом запуску схеми з фіксованим значенням K. Тоді за пірамідальним принципом на кожному рівні визначаються значення K і ∇L_p, що служать основою для корекції ядра на наступному кроці ітерації.

Спочатку з вхідного розмитого зображення ми будуємо піраміду зображень з різню роздільною здатністю від самого маленького розміру до

реального розміру зображення, що обробляється, наприклад, представлений на рисунку 2.2.

Потім запускається алгоритм з початковим ядром розміром 3х3 із заданим одним з простих шаблонів – вертикальна лінія, горизонтальна лінія або гаусівська пляма. В якості універсального варіанта можна вибрати останній – гаусівська пляма.

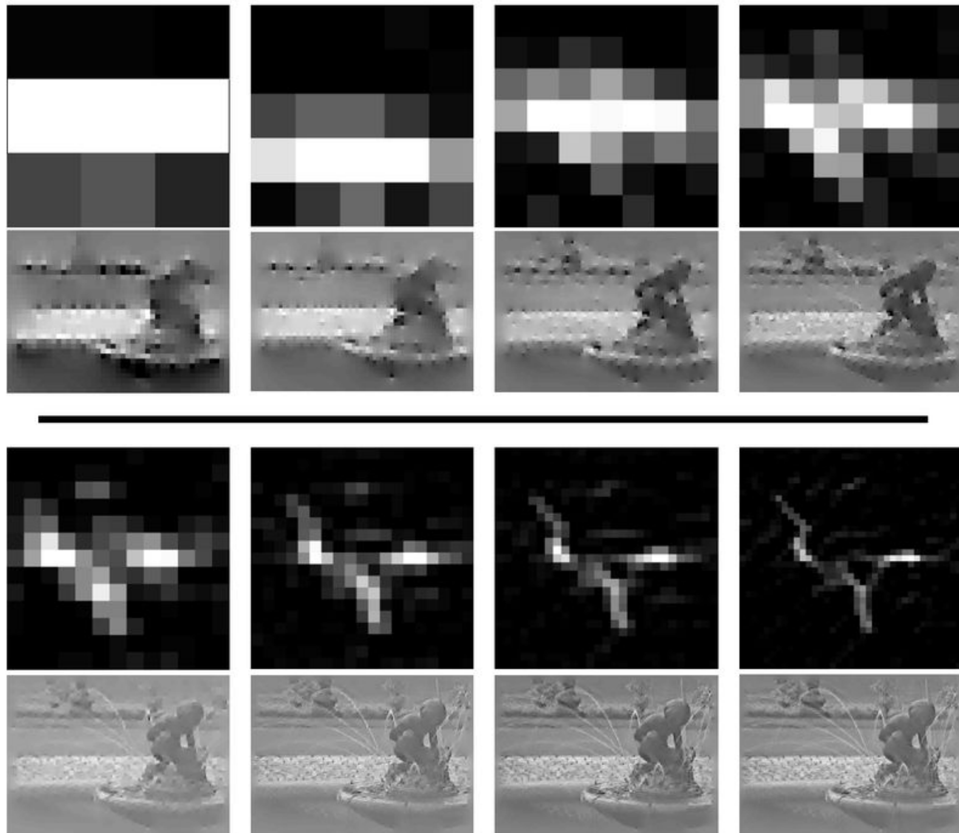


Рисунок 2.2 – Багатомасштабна схема роботи ітераційного алгоритму, який обробляє зображення фонтану.

Використовуючи алгоритм оптимізації, описаний вище покращується оцінка ядра, використовуючи найменший розмір зображення в побудованій піраміді.

Після цього здійснюється уточнення і збільшення ядра, припустимо, до 5x5 пікселів і повторюємо процес вже із зображенням наступного розміру.

Таким чином на кожному кроці поліпшується ядро і в кінцевому результаті отримуємо достатньо точну траєкторію змазу.

Для фінального масштабу, що відповідає реальному розміру оброблювального зображення в кінці ітераційного процесу отримуємо уточнене ядро K.

Перший та третій рядки – це орієнтовані вигляди ядра розмивання на кожному масштабі рівня. Другий та четвертий рядки – оціночне скоректоване зображення на кожному кроці ітерації для уточненого на даному рівні ядра

розмиття. В результаті отримаємо реконструйоване зображення представлене на рисунку 2.3

Рисунок 2.3 – Реконструйоване зображення як результат представленого алгоритму.

Остаточна реконструкція знаходиться за допомогою алгоритму Річардсона-Люсі з кінцевим розрахунковим ядром розмиття.

Незважаючи на те, що більш природно було б запустити багатоступеневу схему виводів, використовуючи повне градієнтне зображення ∇L, на практиці виявилось, що алгоритм виконується краще, якщо менший фрейм вибрати вручну, структура якого насичена границями переходу між об'єктами. Ручний вибір дозволяє користувачеві уникати великих площ насиченості кольорів або їх рівномірності, що погіршує роботу алгоритму через неінформативність початкового фрейму. Приклад вибраного користувачем фрейму показаний на рисунках 2.4.

Рисунок 2.4 – Верхій рисунок – сцена з невеликим розмиттям і обраний користувачем фрейм з написом ієрогліфами, обведений сірим прямокутником. Нижній рисунок –результат роботи алгоритму та отриманим ядром розмиття.

Слід звернути увагу на обраний фрейм обведений сірим прямокутником, який використовується для виявлення ядра розмиття. Він обраний таким чином, щоб було багато деталей зображення і границь переходів, але мало насичених пікселів.

Очевидно, що алгоритм працює набагато швидше на невеликому фреймі , ніж на всьому зображенні. Додатковий параметр – це максимальний розмір ядра розмиття. Розмір розмиття, який виникає в зображенні, коливається в широких межах, від декількох пікселів до сотень. Невеликі розмиття важко виявити, якщо алгоритм на початку ініціалізується з дуже

великим ядром. І навпаки, якщо використовується занадто мале ядро, обрізаються великі розмиття. Отже, для роботи за будь-яких умов, приблизний розмір ядра є необхідним параметром, що задається користувачем. Вивчаючи будь-який розмитий артефакт у зображенні, розмір ядра легко визначається.

На остаток, в розробленій програмі також вимагаємо від користувача вибрати одну з двох початкових оцінок ядра розмиття: горизонтальну або вертикальну лінію. Незважаючи на те, що алгоритм часто можна ініціалізувати в будь-якому стані, це дозволяє визначити правильне ядро високої роздільної здатності і гарантує, що алгоритм починає пошук у правильному напрямку. Правильну ініціалізацію легко визначити, дивлячись на будь-який артефакт ядра розмиття на зображенні.

Процедура багатомасштабного аналізу знаходить оцінку ядра розмивання K, простіше ніж усі можливі інші методи реконструкції зображень. Для відновлення знімка зображень з урахуванням цієї оцінки ядра проведено експеримент з різноманітними методами не «сліпої» деконволюції, зокрема з методами Германа [24], Ніламані [5] та ван Сітера [9]. Хоча багато з цих методів добре працюють у прикладах синтетичного тесту, проте на реальних зображеннях демонструють ряд нелінійностей, відсутніх у синтетичних випадках, таких як негаусівський шум, насичені пікселі, залишкові нелінійності у тоновому масштабі та помилки оцінки в ядрі. Нажаль, для багатьох реальних зображень, більшість методів забезпечують неприйнятний рівень відновлення.

Також метод було протестовано на градієнтах всього зображення ∇B, з фіксованим K. Інтенсивний рисунок було сформовано шляхом реконструкції зображення Пуассона [17]. Крім малої швидкості, було неможливо моделювати зазначені вище нелінійності. Це привело до того, що така реконструкція не є кращою, ніж інші підходи.

Оскільки *L*, переважно, велике, прискорення швидкості обробки, то простіші методи є більш привабливими. Отже, ми відтворюємо латентне кольорове зображення *L* за допомогою алгоритму Річардсона-Люсі (*RL*) [5]. В той часяк *RL* виконувався порівняно з іншими методами, які оцінювалися, лише через кілька хвилин, навіть на великих зображеннях інші, більш складні методи, зайняли години чи дні. RL – це «не-сліпий» алгоритм деконволюції, який інтеративно максимізує функцію правдоподібності Пуасонівської моделі шуму.

Однією з переваг цього більш простого методу є те, що він дає лише невідємні вихідні значення. Було використано реалізацію в Matlab алгоритму оцінки *L*, при заданому *K*, обробки кожного каналу кольору незалежно. При цьому виконувалось 10 *RL*-ітерацій, хоча для більших ядер розмиття може знадобитись і більше ітерацій. Перед запуском RL ми очищаємо *K*, застосовуючи динамічний поріг, виходячи з максимального значення інтенсивності всередині ядра, яке встановлює всі елементи нижче певного значення до нуля, таким чином зменшуючи шум ядра. Над вихідним результатом *RL* проводилась гамма-корекція з $\gamma=2.2$, і його гістограма інтенсивності була суміщена з рівнем *B* (використовуючи функцію *histeq* Matlab), і як наслідок з *L*.

Це детальніше описано в приведену алгоритмі:

Algorithm 1 `Image Deblur`

Require: Blurry image **B**; selected sub-window **P**; maximum blur size ϕ; overall blur direction o ($= 0$ for horiz., $= 1$ for vert.); parameters for prior on $\nabla\mathbf{L}$: $\theta_L = \{\pi_c^s, v_c^s\}$; parameters for prior on **K**: $\theta_K = \{\pi_d, \lambda_d\}$.

Convert **P** to grayscale.

Inverse gamma correct **P** (default $\gamma = 2.2$).

$\nabla\mathbf{P}_x = \mathbf{P} \otimes [1, -1]$. *% Compute gradients in x*

$\nabla\mathbf{P}_y = \mathbf{P} \otimes [1, -1]^T$. *% Compute gradients in y*

$\nabla\mathbf{P} = [\nabla\mathbf{P}_x, \nabla\mathbf{P}_y]$. *% Concatenate gradients*

$S = \lceil -2 \log_2 (3/\phi) \rceil$. *% # of scales, starting with 3×3 kernel*

for $s = 1$ to S **do** *% Loop over scales, starting at coarsest*

  $\nabla\mathbf{P}^s$ =`imresize`($\nabla\mathbf{P}$,$(\frac{1}{\sqrt{2}})^{S-s}$,`'bilinear'`). *% Rescale gradients*

  if (s==1) **then** *% Initial kernel and gradients*

    $\mathbf{K}^s = [0,0,0;1,1,1;0,0,0]/3$. If ($o == 1$), $\mathbf{K}^s = (\mathbf{K}^s)^T$.

    $[\mathbf{K}^s, \nabla\mathbf{L}_p^s]$ = `Inference`($\nabla\mathbf{P}^s$,$\mathbf{K}^s$,$\nabla\mathbf{P}^s$,θ_K^s,θ_L^s), keeping $\mathbf{K}^s$ fixed.

  else *% Upsample estimates from previous scale*

    $\nabla\mathbf{L}_p^s$ = `imresize`($\nabla\mathbf{L}_p^{s-1}$,$\sqrt{2}$,`'bilinear'`).

    $\mathbf{K}^s$ = `imresize`($\mathbf{K}^{s-1}$,$\sqrt{2}$,`'bilinear'`).

  end if

  $[\mathbf{K}^s, \nabla\mathbf{L}_p^s]$ = `Inference`($\nabla\mathbf{P}^s$,$\mathbf{K}^s$,$\nabla\mathbf{L}_p^s$,θ_K^s,θ_L^s). *% Run inference*

end for

Set elements of $\mathbf{K}^S$ that are less than $max(\mathbf{K}^S)/15$ to zero. *% Threshold kernel*

B = `edgetaper`(**B**,$\mathbf{K}^S$). *% Reduce edge ringing*

L = `deconvlucy`(**B**,$\mathbf{K}^S$,10). *% Run RL for 10 iterations*

Gamma correct **L** (default $\gamma = 2.2$).

Histogram match **L** to **B** using `histeq`.

Output: **L**, $\mathbf{K}^S$.

Було проведено експерименти, щоб переконатися, що головним чином розмиті зображення внаслідок зміщення камери на відміну від інших рухів, таких як обертання в площині. З цією метою 8 людей сфотографували білу дошку, яка мала маленькі чорні крапки в кожному кутку. При цьому використовувалась витримка затвора 1 секунда. На рисунку 2.5 показані отримані точки з випадкової вибірки зображень, сфотографованих різними людьми.

Точки в кожному кутку показують ядро розмивання, локальне до кожної частини зображення. Зразки розмивання дуже схожі, підтверджуючи припущення, що просторово-інваріантне розмивання є невеликим в площині обертання.

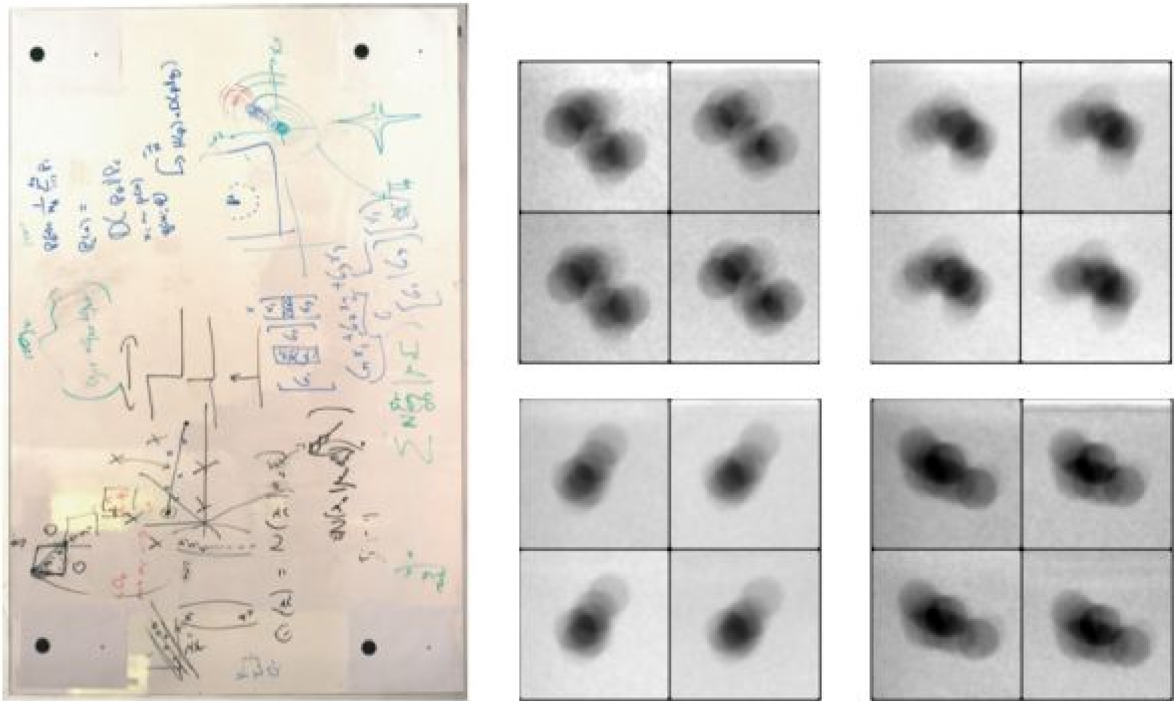


Рисунок 2.5 – Зліва: тестова сцена з дошкою на якій у кожному куті намальовані чорні крапки. Праворуч: крапки з кутів дошки на фотографіях зробленими різними людьми.

Застосувавши алгоритм до ряду реальних зображень з різними ступенями розмиття і насичення. Для кожного показао розмитий образ, а потім – результат роботи алгоритму поряд із оціночним ядром.

Час роботи алгоритму залежить від розміру фрейма, обраного користувачем. З мінімальним практичним розміром 128 × 128 пікселів на даний момент займає робота алгоритму складає 10 хвилин при реалізації в Matlab.

Для виправлення зображення з *N*-пікселів час виконання є *O(N logN)* завдяки використання ШПФ для виконання операцій згортки. Відповідні більші за розміром фрейми будуть опрацьовуватись теж в приємлемих часових межах. Очікується, що оптимізовані версії алгоритму будуть значно покращені і швидші.

На рисунку 2.6 представлено реальне зображення, з незначним розмиттям, яке добре тестує представлений алгоритмом. Виділений

прямокутник показує фрейм, який використовується для обраного ядра розмиття, вибраний таким чином щоб мати багато деталей зображення, але мало насичених пікселів. Результуюче визначене алгоритмом ядро відображаються в куті зображення.

Рисунок 2.6 – Верхній рисунок – сцена зі складними рухами. Хоча саме переміщення камери маленьке, дитина рухаються і особливо рука. Нижній рисунок – результат роботи алгоритму. Обличчя і сорочка стали більш рікими, але рука залишається розмитою, її рух не моделюється за нашим алгоритмом.

На відміну від існуючих «сліпих» методів деконволюції представлений алгоритм може обробляти великі, складні розмиття. На рисунках 2.7 і 2.8 показано як алгоритм успішно визначає великі ядра розмивання.

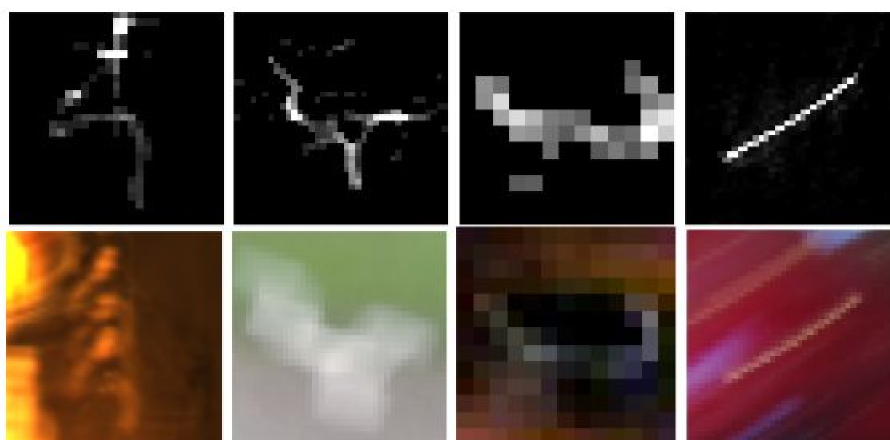
Рисунок 2.7 – Верхій – сцена з великим розмиттям. Нижній – результат роботи алгоритму.

Рисунок 2.8 – Верхній – розмита фотографія. Нижній результат роботи алгоритму. Чітка деталь шпалер тепер видно.

На рисунках 2.7 і 2.8 показано, як алгоритм успішно реконструює великі розмиті ядра. На рисунку 2.9 приведено зображення із складним розмиттям з трьохлистним розмиттям розміром 30 пікселів, яке потім реконструюється.

Рисунок 2.9 – Фотографія фонтану зі складним трьохлистним розмиттям та її відновлення.

Часом істинне ядро розмивання іноді визначається на зображенні за траєкторією точкового джерела світла, трансформованого розмиттям. Це дає можливість порівняти знайдене розмиття з істинним ядром. На рисунку 2.10 показано чотири такі структури зображення, а також визначені відповідні ядра розмиття.



Рисунку 2.10 – Фрейми чотирьох зображень з точковими джерелами світла та відповідні ним ядра розмиття.

Також здійснено порівняння алгоритму з існуючими методами «сліпої» деконволюції, в середовищі Matlab, виконавши процедури відновлення

розмиття, що реалізують методи методів Біггса і Ендрюса [26] та Янссона [27]. На основі повторної схеми Річардсона-Лючі, ці методи також оцінюють ядро розмивання. Результати цього алгоритму, що застосовуються до сцен фонтану та кафе, показані на рисунку 2.11 гірші в порівнянні з отриманим реконструйованим зображенням представленого алгоритму, показаного на рисунках 2.9 і 2.12.

Рисунок 2.11: Базові експерименти з використанням сліпої деконволюції в середовищі Matlab з використанням алгоритмів відновлення знімків фонтану (зверху) і кафе (внизу). Зображення штучно спотворені гаусівським розмиттям з ядром, подібним за розміром до артефактів розмиття.

Зображення зі значною насиченістю.

Рисунок 2.12 містить великі площі, де не спостерігається справжня інтенсивність, через обмеження динамічного діапазону камери.

Користувальницький фрейм, який використовується для аналізу ядра, повинен уникати великих насичених областей. Незважаючи на те, що знімок зображень має деякі артефакти поблизу насичених областей, ненасичені регіони все ще можуть бути відновлені.

Рисунок 2.12 – Верхня частина: розмита сцена з сильною насиченістю, знята з 1-но секундною експозицією. Внизу: вивід результату алгоритму відновлення.

Таким чином, остаточний алгоритм реконструкції зображення можна формалізувати у вигляді схеми представленої на рисунку 2.13.

Запропонований метод дозволяє видалити ефектів від зміщення камери при фотографуванні. На перший погляд ця проблема здається важко вирішуваною. Проте в роботі показано, що, застосовуючи натуральні зображення та вдосконалених статистичних методів, все-таки можна отримати прийнятні результати. Такий підхід може виявитися корисним в інших проблемах цифрових фотографій та зображень. Значна частина зусиль зосереджена на оцінці ядра, яке покращується в описаному ітераційному процесі. Результати нашого методу часто містять артефакти. Найбільш виразно, ефекти «дзвону» виникають біля насичених областей та районів значного руху об'єкта.

Рисунок 2.13 – Структурна схема алгоритму деконволюції розмитих зображень.

При цьому є значні можливості для поліпшення представленого алгоритму, застосовуючи сучасні статистичні методи для проблеми деконволюції та використовуючи поєднання різних підходів до нейтралізації ефектів розмивання.

РОЗДІЛ 3
ВИБІР ТА НАЛАШТУВАННЯ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СИСТЕМИ ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ

3.1. Вибір мови програмування

C++ – мова програмування високого рівня з підтримкою декількох парадигм програмування: об’єктно-орієнтованої, узагальненої та процедурної. Розроблена Б’ярном Страуструпом в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) у 1979 році та початково отримала назву “Сі з класами”. Згодом Страуструп перейменував мову у C++ у 1983 р. Базується на мові С. Визначена стандартом ISO/IEC 14882:2003.

У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відеоігри. С++ суттєво вплинула на інші, популярні сьогодні, мови програмування: С# та Java [28].

C# – об’єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft).

Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов С++, Delphi, Модула і Smalltalk – С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від С++)[29].

Java – об'єктно-орієнтована мова програмування, випущена компанією Sun Microsystems у 1995 році як основний компонент платформи Java. Зараз мовою займається компанія Oracle, яка придбала Sun Microsystems у 2009 році. Синтаксис мови багато в чому схожий на C та C++. У офіційній реалізації, Java програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Oracle надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням [30]. За необхідності таких дій java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++, що розроблялась компанією Microsoft. Роботу над J++ було зупинено через судовий позов компанії Sun Microsystems, оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі Microsoft .NET випустило J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C#, стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java[31].

Для реалізації ПЗ для відновлення зображень була вибрана мова програмування Java.

В індексі TIOBE мови програмування Java і C тривалий час поділяють перше і друге місця, а завдяки недавньому випуску загальнодоступною версією JDK 8 ситуація навряд чи зміниться.

Можна стверджувати що Java найкраща платформа для розробки майже будь-яких додатків і на це є декілька причин.

Java Core API складається з колекції надійних, стійких і зрозумілих бібліотек. Хоча багато хто скаржиться на недостатність функцій у цій галузі (маючи на увазі Google Guava або Apache Commons), вони забувають, що в основі цих розширень лежить Java Core API. Знову-таки, недозволена розкіш для C++.

Java з самого початку могла похвалитися відмінною багатопоточністю і моделлю пам'яті, виправленої лише одного разу в JDK 1.5 в 2004 році, яка сформувала міцну базу для нових моделей на основі акторів (Fork / JOIN і так далі).

Не можна обійти увагою JVM – вона дозволила стільком мовам працювати на широкому ряді апаратних платформ.
І, зрозуміло, в позитивних сторонах JVM є заслуга байткоду. Байткод являє собою незалежну від постачальника абстракцію машинного коду – прогнозовану, яку можна генерувати, обробляти і видозмінювати за допомогою різних технологій. Нещодавно опублікували пост, в якому автор показує, як можна використовувати трансформації байт-коду для емуляції LINQ на Java.

П'ятнадцять років тому розробка ПЗ відбувалася інакше. Можна писати асемблер або C-програми за допомогою редактора vi або Notepad. Але коли ишеться складна корпоративна Java-програма, варто скористатися IDE.
Незважаючи на недоліки зворотної сумісності, все одно вражає, як довго існують мова Java, JVM і JDK без впровадження значних регресій зворотної сумісності. Єдине, що спадає на думку – введення ключових слів assert і enum.

Введення Java 8 Streams API, лямда-виразів, методів за замовчуванням, узагальнень, enums і багатьох інших функцій, не порушуючи роботу системи. Java добре спроектована і зріла платформа, яка ніколи не припиняла оновлюватися. Реліз Java 8 дозволив змінити розробку ПЗ в кращу сторону.

Хоча, в основному, коли йдеться про обробку зображень в 85% для створення програми вибирається C/C++, але не слід недооцінювати інші мови та фактори, що сприяють вибрати не C++. Перший фактор – знайти людей, які знають Java чи Python в рази легше ніж C++. Другий – для Java і Python також є створені ефективні прив’язки до функцій написаних на C. Наприклад такі прив’язки реалізовує бібліотека OpenCV (яка є нативною). Також є спроби створення бібліотек для роботи з зображеннями без сішних прив’язок на pure Java. Однією з таких спроб є бібліотека BoofCV.

Оскільки вибрана мова не є типовою для вибраної області програмування, то в результаті виконання ПЗ необхідно буде провести дослідницьку роботу, чи взагалі обробка зображень на Java доцільна?

3.2. Вибір середовища розробки

Для програмування на JAVA в основному використовують три найвідоміші IDE: NetBeans, Eclipse і IntelliJ IDEA.

NetBeans IDE – вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній конфігурації. Середовище розробки NetBeans за умовчанням підтримує розробку для платформ J2SE і J2EE.

Поширюється у сирцевих текстах під ліцензіями GPLv2 і CDDL. Проект NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun – Oracle, проте розробка NetBeans

ведеться незалежно співтовариством розробників (NetBeans Community) і компанією NetBeans.Org.

NetBeans IDE доступна для платформ Microsoft Windows, GNU/Linux, FreeBSD, і Solaris (як SPARC, так x86). Для інших платформ доступна можливість зібрати NetBeans самостійно із сирцевих текстів.

За якістю і можливостям останні версії NetBeans IDE змагаються з найкращим інтегрованими середовищами розробки для мови Java, підтримуючи рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення мовних конструкцій на льоту, шаблони коду та інше.

Розробка середовища NetBeans почалася в 1996 під назвою Xelfi (гра букв на основі Delphi), як проект студентів зі створення Java IDE під керівництвом факультету математики і фізики Карлова Університету в Празі. У 1997 році Роман Станек сформував компанію навколо проекту і став випускати комерційні версії середовища NetBeans до передачі всіх прав на IDE корпорації Sun Microsystems в 1999 році. Sun відкрила сирцеві коди середовища розробки NetBeans IDE в червні наступного року. Відтоді спільнота NetBeans постійно розвивається і росте завдяки людям і компаніям, що використовують і підтримує проект.

NetBeans IDE 6.0, створена на основі попередньої версії 5.5.1, надала гнучку підтримку створення модулів для IDE і інтернет-застосунків, заснованих на платформі NetBeans, новий дизайнер користувацьких інтерфейсів (відомий під назвою “Проект Matisse”), нову і перероблену підтримку системи управління версіями CVS, підтримку Weblogic 9 і JBoss 4, і багато покращень в редакторі. NetBeans 6.0 поставляється в складі дистрибутивів Ubuntu 8.04 і Debian.

NetBeans IDE 6.5, випущена в листопаді 2008 року, розширює можливості Java EE (включаючи підтримку Java Persistence, EJB 3 та JAX-WS). Додатково, NetBeans Enterprise Pack підтримує розробку застосунків Java EE 5 Enterprise, включаючи візуальні засоби SOA, засоби для роботи з

XML schema, роботу з веб-сервісами (для BPEL), і моделювання на мові UML. Збірка NetBeans IDE Bundle for C/C++ підтримує проекти на мовах C/C++.

NetBeans 7.0, що вийшла у квітні 2011, реалізувала підтримку розробки застосунків з використанням попередньої версії JDK7, були додані засоби для інтеграції з Oracle WebLogic Server 11g і забезпечена підтримка Oracle Database, GlassFish Server Open Source Edition 3.1 і Oracle GlassFish Server 3.1. Версія 7.0 вилучила зі складу модулі з реалізацією засобів розробки мови Ruby і MVC-фреймворка Ruby on Rails. В якості причини названа низька популярність NetBeans серед розробників мовою Ruby.

За заявою Oracle NetBeans IDE 7.1, що вийшов у грудні 2011, став першим середовищем розробки, який повною мірою підтримує останні варіанти специфікацій і стандартів на платформу Java, включаючи повну підтримку циклу розробки з використанням JavaFX і JDK7. Основними нововведеннями NetBeans 7.1 є забезпечення повноцінної підтримки розробки з використанням JavaFX 2.0, значне розширення можливостей Swing GUI Builder, підтримка CSS3, нові інструменти для візуального зневадження інтерфейсу застосунків на базі Swing і JavaFX, інтеграція підтримки Git, додані засоби для інтеграції з Oracle WebLogic Server 12c.

У випуску 7.4 у жовтні 2013 продовжено розвиток засобів для розробки з використанням технологій HTML5, додана підтримка створення гібридних HTML5-застосунків для платформ Android і Apple iOS з використанням фреймворку Apache Cordova, реалізовані засоби використання HTML5 в проектах Java EE і PHP, представлена експериментальна підтримка майбутнього випуску JDK8.

NetBeans 8 вийшов 18 березня 2014. У випуску реалізовані засоби для розробки з використанням Java SE 8, Java SE Embedded 8 і Java ME Embedded 8, розширена підтримка Maven і Java EE з PrimeFaces, додані нові інструменти для HTML5 і, зокрема, фреймворк AngularJS, покращена підтримка PHP (підтримка системи unit-тестування Nette Tester і аналізатора

коду PHP-CS-Fixer; поліпшення підтримки Twig, Latte, Neon) і C/C++ (зокрема додана консоль зневаджувача GDB).

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення. Розробляється і підтримується Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для розробників на мові Java, засоби для управління сирцевими кодами, візуальні побудовники GUI тощо. Написаний в основному на Java, може бути використаний для розробки застосунків на Java і, за допомогою різних плаґінів, на інших мовах програмування, включаючи Ada, C, C++, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з Ruby on Rails), Scala, Clojure та Scheme. Середовища розробки зокрема включають Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C++, Eclipse JDT для Java, Eclipse PDT для PHP.

Випущена на умовах Eclipse Public License, Eclipse є вільним програмним забезпеченням. Він став одним з перших IDE під GNU Classpath.
IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії "Community Edition" і повнофункціональної комерційної версії "Ultimate Edition", для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Вихідний код Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні файли підготовлені для Linux, Mac OS X і Windows.

Перша версія IntelliJ IDEA з’явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу, що дозволяла програмістам швидко реорганізовувати сирцевий код програм. Дизайн середовища орієнтовано на продуктивність праці програмістів, дозволяючи їм сконцентруватися на розробці функціональності, тоді як IntelliJ IDEA бере на себе виконання рутинних операцій.

Починаючи з шостої версії продукту IntelliJ IDEA надає інтегрований інструментарій для розробки графічного користувацького інтерфейсу.
З версії 9.0 є безплатний варіант Community Edition з відкритими кодами. Сирцеві коди відкритої версії IntelliJ IDEA Community Edition поширюються рамках ліцензії Apache 2.0. Бінарні пакунки підготовлені для Linux, Mac OS X і Windows.

До складу IntelliJ IDEA включені напрацювання, створені в результаті спільної роботи з компанією Google, яка використовувала IntelliJ IDEA в якості базису для своєї нового відкритого середовища розробки Android Studio. Завдяки співпраці істотно розширені штатні можливості IntelliJ IDEA з розробки застосунків для платформи Android.

Community версія середовища IntelliJ IDEA підтримує інструменти для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven і Ant, мови програмування Java, Java ME, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Комерційна версія “Ultimate Edition” відрізняється наявністю підтримки додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду, можливістю роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Java Web framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

Для розробки програми “WienerDeblur” було обрано IntelliJ IDEA.

3.3. Огляд засобів створення користувацьких інтерфейсів на Java

Ситуація з GUI фреймворками у світі Java дещо заплутана. Тому було необхідно провести аналіз й вибрати підходящий фреймвок.

AWT (Abstract Window Toolkit) був першим GUI фреймворком. Ідея була правильна – AWT використовує нативні контроли, тобто, вони виглядають і фізично є рідними, незалежно від того, де ви запускаєте свій додаток. На жаль, виявилося, що загальних для різних оточень контролів мало і писати кросплатформні нативні інтерфейси так, щоб нічого не поповзло і не роз'їхалася, дуже складно.

Тому на зміну AWT прийшов Swing. Swing – інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. Це частина бібліотеки базових класів Java (JFC, Java Foundation Classes). Swing розробляли для забезпечення функціональнішого набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у ранішого інструментарію AWT. Компоненти Swing підтримують специфічні look-and-feel модулі, що динамічно підключаються. Завдяки ним можлива емуляція графічного інтерфейсу платформи (тобто до компоненту можна динамічно підключити інші, специфічні для даної операційної системи вигляд і поведінку). Основним недоліком таких компонентів є відносно повільна робота, хоча останнім часом це не вдалося підтвердити через зростання потужності персональних комп'ютерів. Позитивна сторона – універсальність інтерфейсу створених програм на всіх платформах.

На початку існування Java класів Swing не було взагалі. Через слабкі місця в AWT (початковій GUI системі Java) було створено Swing. AWT визначає базовий набір елементів керування, вікон та діалогів, які підтримують придатний до використання, але обмежений у можливостях графічний інтерфейс. Однією з причин обмеженості AWT є те, що AWT перетворює свої візуальні компоненти у відповідні їм еквіваленти, що не

залежать від платформи, які називаються рівноправними компонентами. Це означає, що зовнішній вигляд компонентів визначається платформою, а не закладається в Java. Оскільки компоненти AWT використовують "рідні" ресурси коду, вони називаються ваговитими.

Використання "рідних" рівноправних компонентів породжує деякі проблеми. По-перше, у зв`язку із різницею, що існує між операційними системами, компонент може виглядати або навіть вести себе по-різному на різноманітних платформах. Така мінливість суперечила філософії Java: "написане один раз, працює скрізь". По-друге, зовнішній вигляд кожного компонента був фіксованим (оскільки усе залежало від платформи), і це неможливо було змінити (принаймні, це важко було зробити). По-третє, використання ваговитих компонентів тягнуло за собою появу нових обмежень. Наприклад, ваговитий компонент завжди має прямокутну форму і є непрозорим.

Незабаром після появи початкової версії Java, стало очевидним, що обмеження, властиві AWT, були настільки незручними, що потрібно було знайти кращий підхід. У результаті з'явились класи Swing як частина бібліотеки базових класів Java (JFC). В 1997 році вони були включені до Java 1.1 у вигляді окремої бібліотеки. А починаючи з версії Java 1.2, класи Swing (а також усі останні, що входили до JFC) стали повністю інтегрованимими у Java.

Swing використовує форми, створювані AWT, на яких він своїми засобами малює контроли. Працює це повільніше, але зате UI стає набагато більш переносимим. Swing пропонує на вибір програмісту безліч Look & Feel, завдяки яким можна зробити або так, щоб додаток виглядав і вів себе однаково як під Windows, так і під Linux, або щоб додаток був дуже схожим на нативний незалежно від того, де його запускають. У першому випадку додаток простіше налагоджувати, у другому – він стає приємніший для користувачів. До речі, спочатку Swing був зроблений розробниками з Netscape.

SWT (Standard Widget Toolkit) – фреймворк, написаний в IBM і використовуваний в Eclipse. Як і в AWT, використовуються нативні контроли. SWT не входить в JDK і використовує JNI, тому не дуже відповідає ідеології Java "написано одного разу, працює скрізь". При дуже сильному бажанні можна запакувати в пакет реалізацію SWT для всіх платформ, і тоді додаток ніби як навіть стане переносимим, але тільки до тих пір, поки не з'явиться якась нова операційна система або архітектура процесора.

JavaFX активно розробляється в Oracle і позиціонується, як швидка заміна Swing. Ідеологічно JavaFX схожий на Swing, тобто, контроли не нативні. Серед цікавих особливостей JavaFX слід зазначити хардверне прискорення, створення GUI за допомогою CSS і XML (FXML), можливість використовувати контроли JavaFX в Swing, а також багато нових гарних контролів, в тому числі для малювання діаграм і 3D. Починаючи з Java 7, JavaFX є частиною JRE/JDK.

NetBeans Platform (не плутати з NetBeans IDE) – працює поверх Swing і JavaFX, надає як би зручніший інтерфейс для роботи з ними, а також всякі додаткові контроли.

Вище говорилося про Look & Feel. Програма буде виглядати наступним чином (див. рисунок 3.1).

JRE під Windows і Linux включають в себе різний набір L&F. Крім того, можна підключити сторонній Look&Feel або навіть написати свій. За замовчуванням використовується L&F Metal, який у всіх ОС і віконних менеджерах виглядає більш-менш однаково. Якщо більше до вподоби круглі кнопочки, то замість Metal можна використовувати Look & Feel Nimbus. При бажанні, щоб додаток було схожим на нативний, то під Linux слід вибрати L&F GTK+ (але проблеми можуть виникнути якщо користувач виокористовує KDE), а під Windows – L&F Windows. Непоганою ідеєю, мабуть, буде передбачити у програмі можливість перемикатися між різними

L&F. З іншого боку, при цьому доведеться тестувати роботу програми з усіма цими L&F.

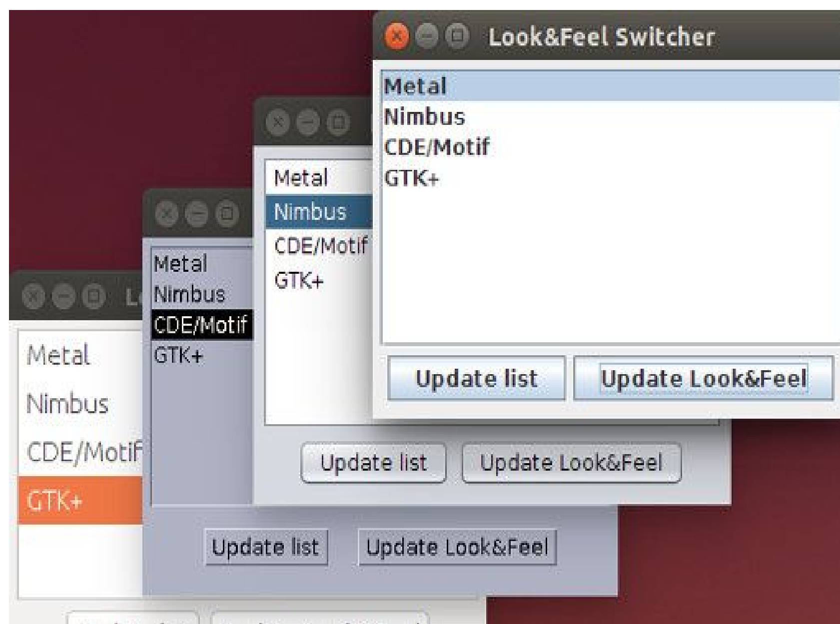


Рисунок 3.1 – Вигляд консолів з різнимим пунктами

Не виключено, що є й інші фреймворки. Найбільш стабільним на сьогоднішній день є Swing, тому дану технологію було було обрано для розробки користувацього інтерфейсу програмного забезпечення.

3.4 Обгрунтування вибору бібліотеки для роботи із зображеннями

OpenCV (англ. Open Source Computer Vision Library, бібліотека комп’ютерного зору з відкритим кодом) – бібліотека функцій та алгоритмів комп’ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об’єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об’єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях.

Бібліотека розроблена Intel і нині підтримується Willow Garage та Itseez. Сирцевий код бібліотеки написаний мовою C++ і поширюється під ліцензією BSD. Біндинги підготовлені для різних мов програмування, таких як Python, Java, Ruby, Matlab, Lua та інших. Може вільно використовуватися в академічних та комерційних цілях.

Офіційно проект OpenCV був запущений у 1999 році за ініціативою Intel Research з ціллю розвивати CPU-ресурсомісткі додатки. Основними вкладниками у проект була Intel's Performance Library Team та певна кількість експертів з чисельної оптимізації у Inter Russia. На перших етапах розвитку OpenCV основними задачами бібліотеки були:

Розвивати дослідження у напрямку комп'ютерного зору, забезпечуючи добре оптимізований та відкритий код бібліотеки.

Поширювати знання у сфері комп'ютерного зору, забезпечуючи загальну інфраструктуру, яку б могли розвивати розробники, таким чином код ставатиме більш легким для сприйняття та обміну.

Розвивати засновані на роботі з комп'ютерним зором комерційні додатки, створюючи не залежну від платформи, оптимізовану та безкоштовну бібліотеку. Для цього використовувалася ліцензія, яка не вимагала від таких комерційних додатків бути відкритими.

Перша альфа-версія OpenCV була оприлюднена на IEEE конференції з комп'ютерного зору й розпізнавання образів у 2000 році, і п'ять бета-версій було випущено у період між 2001 і 2005 роками. Перша версія 1.0 була випущена у 2006 році. У середині 2008 року, OpenCV отримала корпоративну підтримку від Willow Garage і знову перейшла у стадію активної розробки. "пре-релізна" версія 1.1 була випущена у жовтні 2008 року.

Другий великий випуск OpenCV відбувся у жовтні 2009 року. OpenCV 2 включала у себе серйозні зміни у інтерфейсі C++. Ці зміни спрямовані на більш прості, тип-безпечні моделі, додавання нових функцій, і кращу

реалізацію існуючих моделей в плані швидкодії (особливо на багатоядерних системам).

Офіційні релізи надалі відбуваються кожні 6 місяцяві розробкою займається незалежна команда з Росії, яка підтримуються комерційними корпораціями.

У серпні 2012 року, підтримку OpenCV було передано некомерційній організації, OpenCV.org.

BoofCV є Java бібліотекою з відкритим вихідним кодом для додатків реального часу у галузі комп’ютерного зору і робототехніки. Написана з нуля для простоти використання і високої продуктивності, бібліотека часто перевершує навіть бібліотеки, написані на під нативну платформу. Функціональність включає оптимізовані процедури низького рівня для обробки зображень, відстеження паттернів і геометричний комп’ютерний зір. BoofCV був випущений під ліцензією Apache як для академічної так і для комерційного використання.

BoofCV організована в декілька пакетів: обробка зображень, пошук ознак, геометричне бачення, калібрування, розпізнавання, візуалізація і ввід/вивід. Пакет обробки зображень містить часто використовувані функції обробки зображень, які працюють безпосередньо з пікселями. Пакет для пошуку ознак містить алгоритми виділення ознак для використання у операціях вищого рівня. Калібрування містить підпрограми для визначення внутрішніх та зовнішніх параметрів камери. Розпізнавання віповідає за розпізнавання і відстеження складних візуальних об’єктів. Геометричне бачення складається з підпрограм для обробки виділених ознак зображення за допомогою 2D-і 3D-геометрії. Візуалізація має функції для візуалізації і відображення виділених ознак. Ввід/вивід містить основні правила, для зчитування зображень з різних джерел сигналу.

Потрібно також віжзначити зручність у користуванні бібілотекою – просто підключити скомпільований .jar-файл. При чому пакети для різних задач обробки розділені на окремі файли. Тому не приходиться тягнути весь

непотрібний для реалзіації завдання код в свій проект, а лиш використовувати необхідні алгоритми.

В контексті продуктивності в порівнянні з нативною OpenCV розробники надають порівняльну характеристику (рисунок 3.2).

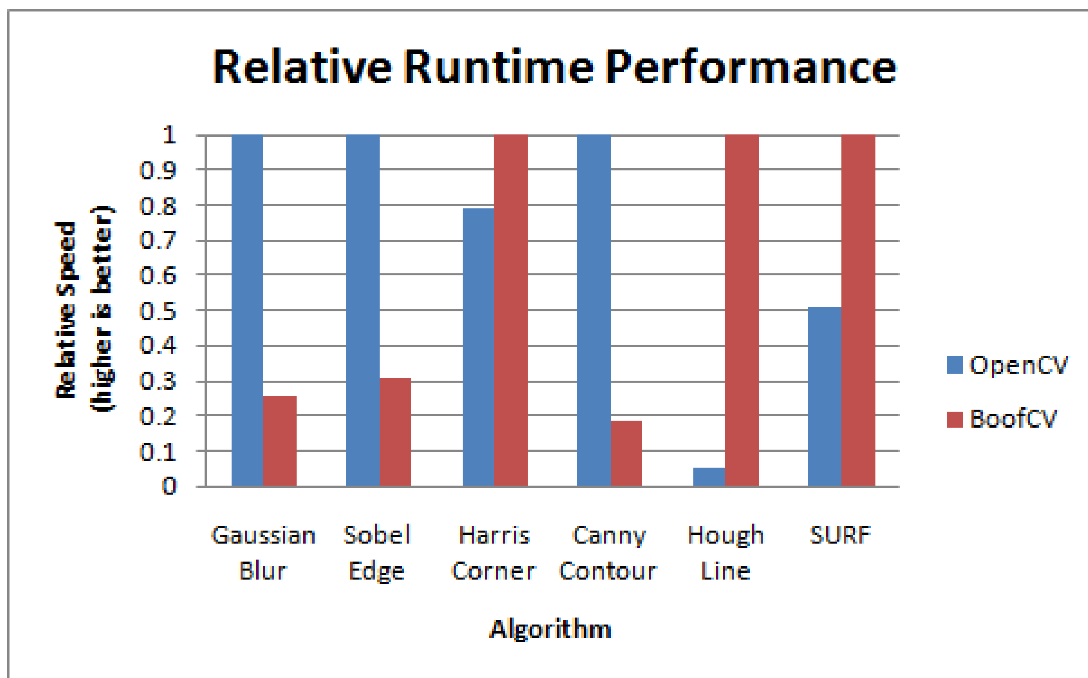


Рисунок 3.2 – Статистика продуктивності BoofCV

Видно що на великій кількості малих операцій OpenCV як нативна бібліотека звичайно краща. Але на малій кількості крупніших операцій BoofCV показує себе продуктивнішою.

РОЗДІЛ 4
РОЗРОБКА СПЕЦІАЛІЗОВАНОЇ КОМПЮТЕРНОЇ СИСТЕМИ ДЛЯ ВІДНОВЛЕННЯ НЕЧІТКИХ ЗОБРАЖЕНЬ

4.1. Проектування програмного забезпечення

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

Діаграми варіантів використання складають модель варіантів використання. Варіант використання – це функціональність системи, яка дозволяє користувачеві отримати якийсь істотний для нього, відчутний та вимірюваний результат. Кожен варіант використання відповідає окремому сервісу, що надається модельованою системою у відповідь на запит користувача, тобто визначає спосіб використання цієї системи.

Варіанти використання найчастіше застосовуються для специфікації зовнішніх вимог до проектованої системи або для специфікації функціональної поведінки вже існуючої системи. Окрім цього, варіанти використання неявно описують типові способи взаємодії користувача з системою, що дозволяють коректно працювати з сервісами, що надаються системою.

Діаграма варіантів використання програми “WienerDeblur” представлена на рисунку 4.1.

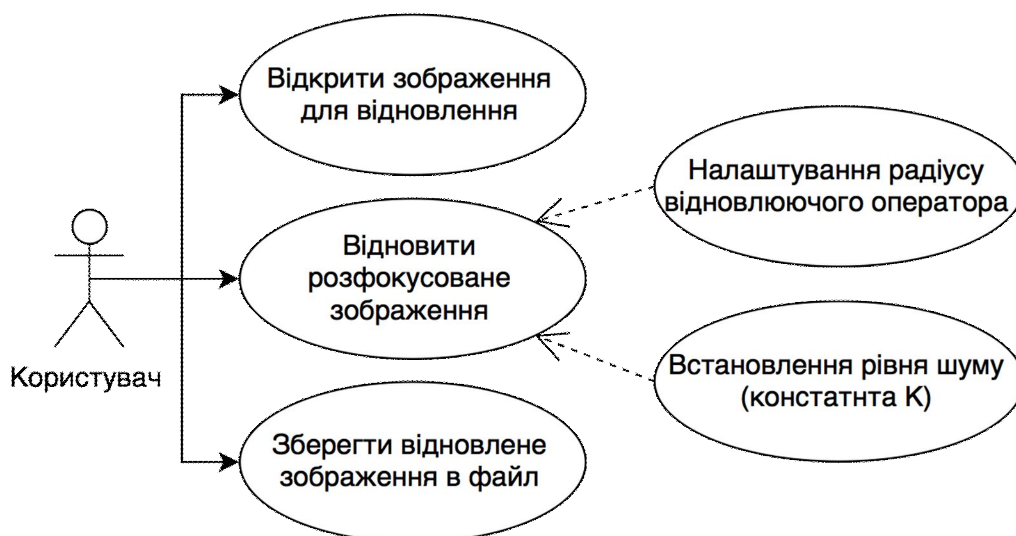


Рисунок 4.1 – Діаграма прецедентів проектованого ПЗ

З діаграми видно, що основними функціональними можливостями програми є:

- відкриття розмазаного зображення;
- збереження відновленого зображення в файл;
- відновдення зображення – основна функція програми.

Під час відновлення зображення, необхідно буде встановити радіус оператора відновлення і рівень шуму на зображенні, для отримання кращої якості реконструкції.

Діаграма класів призначена для надання статичної структури моделі системи в термінології класів об’єктно-орієнтованого програмування. Діаграма класів відображує різні взаємозв’язки між окремими сутностями предметної області, такими як об’єкти й підсистеми, а також описує їхню внутрішню структуру й типи відносин. На даній діаграмі не вказується інформація про часові аспекти функціонування системи.

Клас визначає атрибути і методи набору об’єктів. Всі об’єкти цього класу (екземпляри цього класу) мають спільну поведінку і однаковий набір

атрибутів (кожен з об’єктів має свій власний набір значень).

Діаграма класів для розроблюваної програми містить класи (рисунок 4.2):

- Main;
- DeblurGUI;
- FocusBlur;
- DeconvolutionTool.

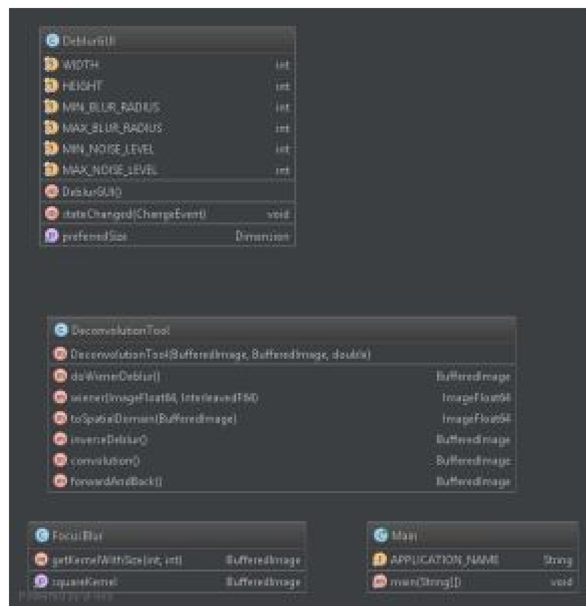


Рисунок 4.2 – Діаграма класів

Main – найпростіший клас в програмі. Він є точкою запуску програми. Виконання починається із запуску спеціального статичного методу main(String[] args). В цьому методі виконується виклик методу showGUI() на новому потоці. showGUI() виконує ініціалізацію користувацього інтерфейсу: створює вікно і задає призначення стандартним кнопкам вікна.

DeblurGUI – реалізує користувацький інтерфейс. Наслідується від JFrame для використання з іншими контролами та розміщення в вікні . Клас відповідає за розташування всіх елементів програми: кнопок відкриття і збереження файлів, полів налаштування радіусу та рівня шуму, вихідного

зображення, оператора змазу та результату відновлення.

FocusBlur має одне поле – radius. Створений для абстрагування зображення оператора змазу. Даний клас генерує чорний прямокутник з білим кругом певного радіусу у вигляді BufferedImage (рисунок 4.3) для використання його при відновленні.

Також даний клас призначений для створення зменшеного квадратного зображення для використання в користувацькому інтерфейсі.
DeconvolutionTool – сутність, що виконує роботу по відновленні зображення.

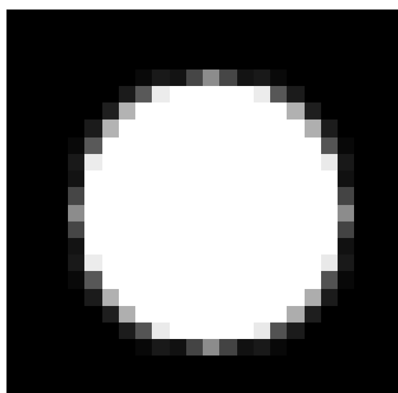


Рисунок 4.3 – Згенерований оператор змазу

Конструктор приймає зображення, яке необхідно відновити, зображення ядра змазу та рівень шуму. З допомогою методу doWienerDeblur() використовується для відновлення зображення.

4.2. Реалізація програмного забезпечення

Для створення користувацького інтерфейсу була використана технологія Java Swing. В класі DeblurGUI використано такі елементи бібліотеки: JPanel, JLabel, JSpinner, JButton. Користувацький інтерфейс складається з кнопок завантаження зображення для обробки та зберігання результату роботи в файл, панелей для відображення зображень –

спотвореного, відновленого і оператора змазу та полів для редагування радісу оператора розмиття та рівня шуму.

Для відслідковування зміни стану контролів використовується паттерн listener. Клас DeblurGUI реалізовує інтерфейс ChangeListener і його метод stateChanged(ChangeEvent e). В при створенні нового контрола, клас підписується на оновлення від нього, і при зміні стану одного з них виконує відповідні операції. При зміні полів радіусу чи рівня шуму викликається метод stateChanged(ChangeEvent e) який перезапускає процес відновлення зображення з новими параметрами.

При натисканні на кнопку відкриття файлу встановлюється поле input, і відображується в панелі з вихідним зображенням. Потім виконується спроба відновлення зі стандартними параметрами.

Для розташування елементів в вікні використовується GridBagLayout і для кожного елементу встановлюється його положення та розімір за допомогою GridBagConstraints. Для розміщення контролів цим методом необхідно для кожного елемента інтерфейсу вказати:

- gridx;
- gridy;
- weightx;
- gridwidth;
- gridheight.

Приклад розміщення елементів інтерфейсу відображено в лістингу 4.1.

Лістинг 4.1 – Розміщення елементів інтерфейсу в GridBagLayout

```
setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.fill = GridBagConstraints.BOTH;
c.gridx = 0;
c.gridy = 0;
c.weightx = 0.1;
c.weighty = 0.05;
add(open, c);
```

Було б зручніше, якби розміщення користувацьких нативних компонентів в Java реалізовувалось декларативною мовою як це зроблено в окремих бібліотеках .Net, Foundation в OSX і iOS, Android тощо. Але на даний момент для десктопних аплікацій відсутня зручна нативна розробка користувацього інтерфейсу, а комерційні реалізації досить дорогі.

Точкою запуску процесу відновлення зображення є виклик методу doDeblur() в DeblurGUI. В цьому методі спочатку створюється екземпляр класу FocusBlur і отримується зображення ядра змазу, яке одночасно і замінюється в інтерфейсі. Потім створюється DeconvolutionTool з вхідними параметрами: вхідне зображення (BufferedImage), ядро змазу (BufferedImage) та рівень шуму. Далі для вихідного зображення присвоюється зображення отримане з методу doWienerDeblur() та оновлюються всі зображення на користувацькому інтерфейсі. Створення та використання DeconvolutionTool продемонстровано в лістингу 4.2.

Лістинг 4.2 – Використання DeconvolutionTool

```
FocusBlur blur = new FocusBlur(radius);
kernel = blur.getSquareKernel();
DeconvolutionTool deconvolutionTool = new
DeconvolutionTool(input,
blur.getKernelWithSize(input.getWidth(), input.getHeight()),
                                    noise);
output = deconvolutionTool.doWienerDeblur();
```

Метод doWienerDeblur() необхіно розглянути детально. Даний метод є ключовим в програмі і виконує роботу по відновленню зображення.
В BoofCV для операцій на зображеннями замість BufferedImage (який є досить “важким” для поелементної роботи над пікселями) використовуються оптимізовані типи, що базуються на одновимірних масивах.

ImageFloat64 – репрезентує зображення, пікселі якого – 64-бітні float. Валідними вважаються значення від 0,0 до 255,0. В даному типі зберігаються значення, що отримуються з BufferedImage. Дані зберігаються в одновимірному масиві data[], але на зовні віддається високорівневий API для

роботи з матрицею зображення:

- get(int x, int y);
- set(int x, int y, double value);
- setData(double[] data);
- unsafe_get(int x, int y);
- unsafe_set(int x, int y, double value).

Сворююється екземпляр типу за допомогою конструктора ImageFloat64(float width, float height).

InterleavedF64 – так як і ImageFloat64 призначений для збереження пікселів зображення в 64-бітному float. Але додатково в даний тип може зберігати кілька каналів для зображення – наприклад значення червоного, зеленого і синього кольорів в для одного пікселя. Для створення екземпляру даного класу окрім кількості пікселів по ширині і висоті потрібно вказати кількість каналів. Дані зберігаються як і в ImageFloat64 в одновимірному масиві data[]. Але надається таке ж високорівневе API:

- get(int x, int y, double[] storage);
- getBand(int x, int y, int band);
- set(int x, int y, double... value);
- setBand(int x, int y, int band, double value).

Для збереження зображень в просторовій області ($g(x,y)$, $h(x,y)$, $f(x,y)$) використовується тип ImageFloat64 в який зображення конвертується з BufferedImage. В частотній області необхідно зберігати два значення – дійсну і комплексну частину. Тому використовуємо InterleavedF64 з двома каналами. Значення $G(u,v)$, $H(u,v)$, $F(u,v)$ зберігаються саме в цьому типі.

Оскільки вся робота з відновлення проводиться в частотній області, то перш за все необхідно перетворити наявні зображення з просторової областів в частотну. Для цієї операції в BoofCV створено окреми тип DiscreteFourierTransform<I, T>, що дозволяє робити операції перетворення з просторової в частотну область і навпаки. Надається API:

- forward(I image, T transform);
- inverse(T transform, I image).

Спосіб перетворення оператора *h(x,y)* демонструє лістинг 4.3.

Лістинг 4.3 – Перетворення в частотну з просторової області і навпаки

```
InterleavedF64   H   =   new   InterleavedF64(kernel.getWidth(), kernel.getHeight(), 2);
ImageFloat64 h = toSpatialDomain(kernel);
dft.forward(h, H);
// do something with H
dft.inverse(H, h);
```

Для відновлення кольорового rgb-зображення необхідно провести відновлення для кожного каналу. Алгоритм приймає на вхід BufferedImage і розбиває його на три сутності типу ImageFloat64 для червоної, зеленої та синьої компонент. Для "витягування" з BufferedImage окремі r, g i b, необхідно використати побітові операції як в лістингу 4.4.

Лістинг 4.4 – Взяття значення кольору для кожного з каналів в rgb

```
int color = input.getRGB(i, j);
int red   = (color & 0xff0000) >> 16;
int green = (color & 0xff00) >> 8;
int blue  =  color & 0xff;
```

Далі створюється частостне представлення для кожної з компонент методом forward(), яке зберігається в InterleavedF64.

Далі вся робота проводиться над частотним представленням зображення. В лістингу 4.5 представлено реалізацію формули (1.8).

Лістинг 4.5 – Реалізація вінерівської фільтрації

```
for (int i = 0; i < F.width; i++) {
     for(int j = 0; j < F.height; j++) {
          double gg = G.getBand(i, j, 0);
          double gi = G.getBand(i, j, 1);
          double hh = H.getBand(i, j, 0);
          double hi = H.getBand(i, j, 1);
          double h22  = H22.getBand(i, j, 0);
          double h22i = H22.getBand(i, j, 1);
```

```
            double mult1Re = divideComplex(1, 0.0, hh, hi)[0];
            double mult1Im = divideComplex(1, 0.0, hh, hi)[1];
            double mult2Re = divideComplex(h22, h22i, h22 + K,
h22)[0];
            double mult2Im = divideComplex(h22, h22i, h22 + K,
h22)[1];

            double multRe = multiplyComplex(mult1Re, mult1Im,
mult2Re, mult2Im)[0];
            double multIm = multiplyComplex(mult1Re, mult1Im,
mult2Re, mult2Im)[1];
            F.setBand(i, j, 0, multiplyComplex(multRe, multIm, gg,
gi)[0]);
            F.setBand(i, j, 1, multiplyComplex(multRe, multIm, gg,
gi)[1]);
      }
}
```

Слід також відзначити що добуток оператора *H(u,v)* і його комплексного спряження виконується вбудованою в бібліотеку функцією CirculantTracker.elementMultConjB(). Реалізація виглягає дещо складнішою, в порівнянні з інверсним фільтром (лістинг 4.6).

Лістинг 4.6 – Реалізація інверсного фільтра

```
for (int i = 0; i < F.width; i++) {
      for(int j = 0; j < F.height; j++) {
            double G0 = G.getBand(i, j, 0);
            double G1 = G.getBand(i, j, 1);
            double H0 = H.getBand(i, j, 0);
            double H1 = H.getBand(i, j, 1);
            double F0 = (G0 * H0 + G1 * H1) / (H0 * H0 + H1 * H1);
            double F1 = (H0 * G1 + G0 * H1) / (H0 * H0 + H1 * H1);
            F.setBand(i, j, 0, F0);
            F.setBand(i, j, 1, F1);
      }
}
```

Після виконання фільтра для всі каналів кольору, зображення перетворюються з частостної області в просторову методом inverse(). Далі з трьох зображень для червоного, зеленого і синього каналів формуєсья одне BufferedImage. Формування з трьох канлів одного зображення зображено в лістингу 4.7.

Лістинг 4.7 – Формування кольорового зображення з трьох каналів

```
int red   = (int)wienerOutRed.get(i, j);
int green = (int)wienerOutGreen.get(i, j);
int blue  = (int)wienerOutBlue.get(i, j);
int addBright = 0;
int RGB = (red   + << 16) |
          (green +) << 8)|
           blue;
result.setRGB(i, j, RGB);
```

Нове зображення виводиться на екран і його можна зберегти чи перерахувати з іншими параметрами.

Окремо необхідно описати представлення рівня шуму в програмі. Для користувача буде зручно задавати рівень шуму в відсотках – від повністю безшумного до цілком нерозбірливого. Однак в реальному зображенні навіть 0.0005 частина шуму призводить до сильної зашумленості при обробці в частотній області. Тому щоб конвертувати користувацький ввід в справжню кількість шуму використовується вираз K = Math.pow(1.07, noiseLevel)/10e-4.

4.3. Тестування програмного забезпечення

Інтерфейс програми WienerDeblur зображено на рисунку 4.4.

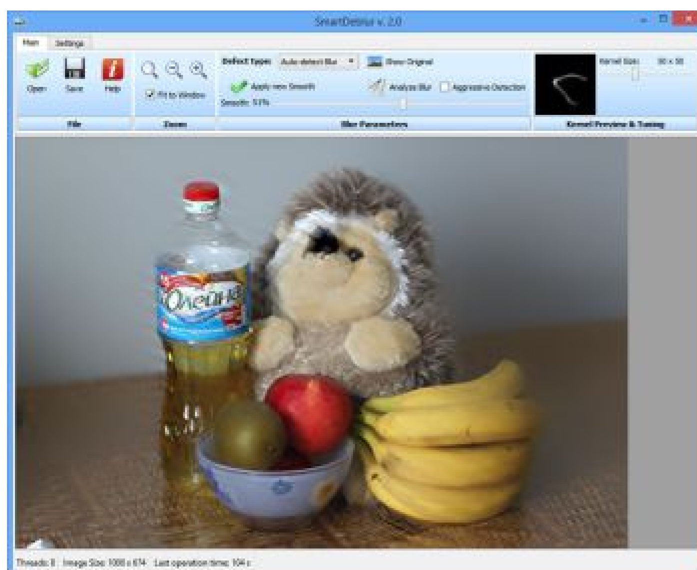


Рисунок 4.4 – Інтерфейс програми

Основними характеристиками продукту для відновлення зображень є його: точність відновлення; швидкодія.

Заміри швидкодії вимірювались для функції, яка безпосередньо виконує відновлення для одного каналу зображення – wiener(), та для методу doWienerDeblur(), який повертає повне відновлене зображення. Для тесту було обрано 5 квадратних зображень розмірами:

- 300*300;
- 400*400;
- 500*500;
- 600*600;
- 700*700.

Кожен замір проводився три рази, за результат взято середнє арифметичне. На рисунку 4.5 зображено графік залежності часу виконання обробки для одного каналу від розмірності зображення. Тест проводився на машині з параметрами:

- процесор – Intel Core i3 2.0ГГц;
- 4 Гб оперативної пам’яті.
-

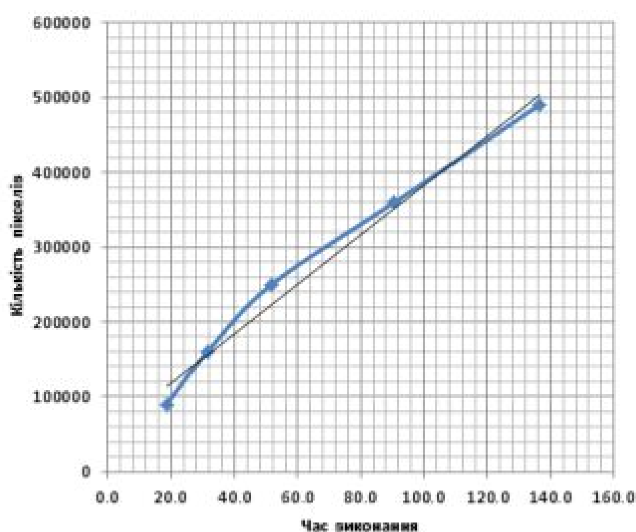


Рисунок 4.5 – Залежність часу виконання алгоритму відновлення одного каналу від розміру зображення

На рисунку 4.6 – залежність часу виконання методу doWienerDeblur().

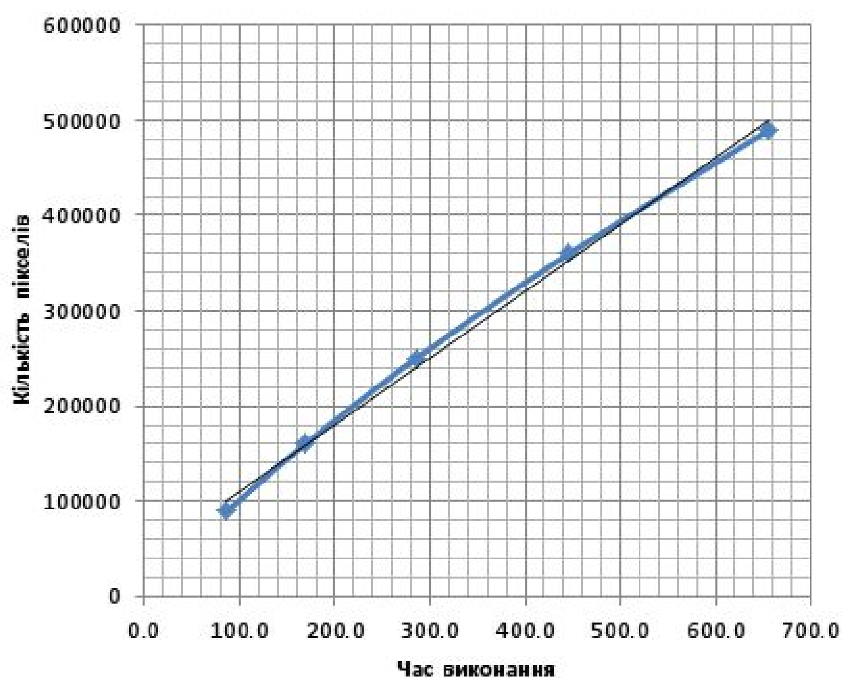


Рисунок 4.6 – Залежність часу виконання алгоритму відновлення повністю зображення від його розміру

На обох графіках прослідковується лінійна залежність. Можна приблизно порахувати скільки часу займе обробка зображення більшого розміру. Наприклад для розміру 4000*3000 приблизний час виконання становитиме: 4000*3000 = 12000000/500000 = 24; 500000 пікселів обробляється на графіку 650 мс, отже для великого зображення це буде приблизно 24 * 650 = 15600 мс. Досить довго, але потрібно розуміти що:

- це дійсно велике зображення, і тут буде виникати питання з пам’яттю на комп’ютері;
- існують способи оптимізувати алгоритм хоча б на 30%;
- це все ж таки Java і gc забирає дорогоцінний час.

Для оцінки якості відновлення візьмемо зображення на рисунку 4.7.

Рисунок 4.7 – Вхідне зображення

Для початку спробуємо зробити його чітким за допомогою інверсного фільтра. Результат приведений на рисунку 4.8.

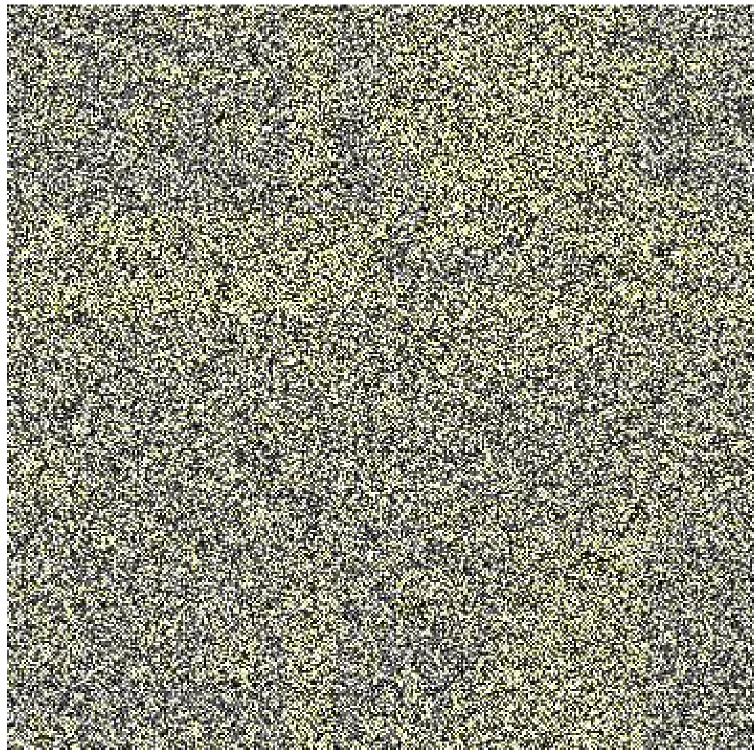


Рисунок 4.8 – Результат роботи інверсного фільтра

Для роботи з даним фільтром необхідна додаткова обробка зображення ще кількома фільтрами, тому даний результат хоч і правильний але зовсім неприйнятний.

Далі відновимо рис. 4.7 вінерівським фільтром з радіусом 15.2. Результат роботи приведено на рисунку 4.9

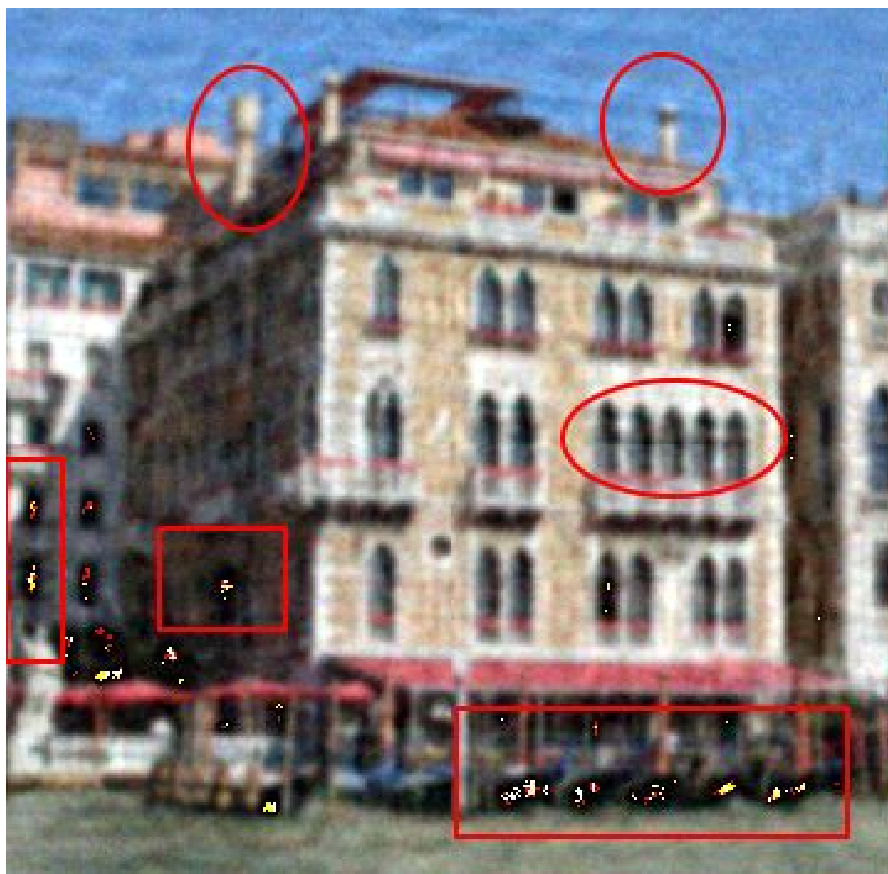


Рисунок 4.9 – Результат відновлення.

Після обробки зразу стають видні деталі, обведені овалами – комини і форми вікон. Але при використанні даного фільтра інколи проявляються артефакти (в прямокутниках), причиною яких є особливості реалізації зворотнього перетворення Фур'є в бібліотеці BoofCV.

Також слід відзначити що даний спосіб відновлення не допоможе при змазі іншого типу, наприклад змазі під час руху як на рисунку 4.10 .

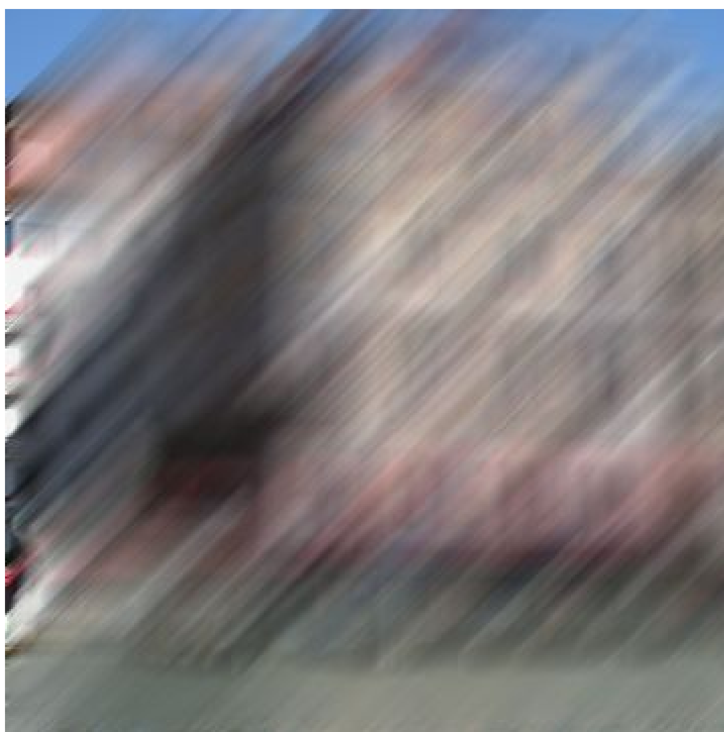


Рисунок 4.10 – Змаз при русі по прямій.

В даному випадку слід використовувати інше ядро змазу до відновлення. Результат спроби відновлення продемонстровано на рисунку 4.11.

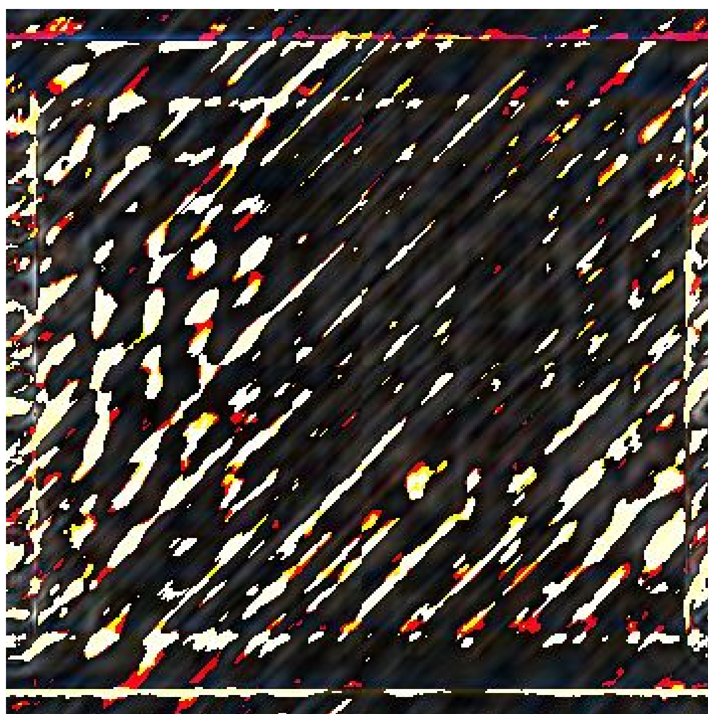


Рисунок 4.11 – Відновлення при іншому типі змазу

Тепер спробуємо застосувати інший оператор змазу, щоб побудувати відновлення по ньому (рисунок 4.12).

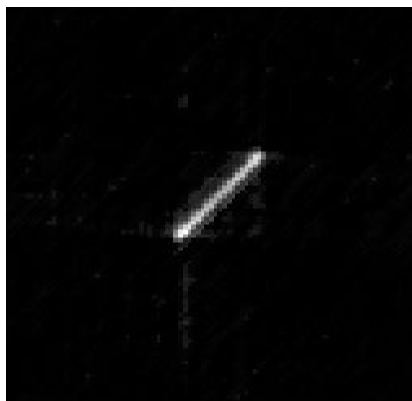


Рисунок 4.12 – Оператор змазу під час руху

При використанні такого оператора на зображенні (рисунок 4.13) зразу стає видно деталі.

Рисунок 4.13 – Відновлене зображення

Необхідно зробити висновок, що для кожного випадку розмиття необхідно підбирати відповідні ядра змазу для отримання адекватного результату.

ВИСНОВКИ

В процесі виконання роботи було успішно розроблено програмний продукт для відновлення розмитих зображень. Під час розробки було Отримано наступні результати.

1. Проаналізовано типи розмиття зображень та основних причин їх виникнення при здійсненні зйомок різними типами фотокамер.

2. Досліджено існуючі методи відновлення нечітких зображень з аналізом їх можливостей, умов ефективного застосування, їх переваг та недоліків.

3. Проаналізовано методи виявлення та уточнення ядра розмиття зображення, на основі якоговідвувається реконволюція.

4. Обгрунтовано вибір метода Річардсона –Люсі як основу для деконволюції кольорових зображень.

5. Обгрунтовано вибір інтегрованого середовища розробки JavaSE7.

6. Розроблено програмний модуль комп’ютерно-інтегрованої системи відновлення нечітких жображень.

7. Для покращення процесу деконволюції реалізовано фільтри Вінера та Тихонова.

8. Протестовано роботу розробленого програмного модуля на ряді фотографій представлених у 3 розділі роботи та підтверджено ефективність його роботи.

У порівнянні з моєю бакалаврською роботою здійснено удосконалення цільвої функції, на основі якої ітераційно уточнюється ядро розмиття, що дозволило більш якісно та швидко відновлювати кольорові зображення. Також додатково реалізовано метод деконволюції TV prior та для досягнення кращої чіткості реалізовано фільтр Тихонова. У порівнянні з попередньою версією програми швидкість її роботи збільшено в середньому у 2,5 рази при цьому у 1,5 рази зменшено обсяг використання пам’яті. Крім того, стало

можливим обробляти більші за об’ємом пам’яті зображення. Додано більше користувацьких налаштувань та виправлено деякі програмні недоліки.

В загальному робота успішно виконана на основі сучасних теоретичних досягнень в галузі деконволюції зображень з внесенням власного наукового вкладу з практичною реалізацією, що підтверджуюються реальними прикладами застосування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wiener, Norbert (1949). Extrapolation, Interpolation, and Smoothing of Stationary Time Series. New York: Wiley. ISBN 0-262-73005-7.
2. MISKIN, J., AND MACKAY, D. J. C. 2000. Ensemble Learning for Blind Image Separation and Deconvolution. In Adv. in Independent Component Analysis, M. Girolani, Ed. Springer-Verlag.
3. KUNDUR, D., AND HATZINAKOS, D. 1996. Blind image deconvolution. IEEE Signal Processing Magazine 13, 3 (May), 43–64.
4. JALOBEANU, A., BLANC-FRAUD, L., AND ZERUBIA, J. 2002. Estimation of blur and noise parameters in remote sensing. In Proc. of Int. Conf. on Acoustics, Speech and Signal Processing.
5. NEELAMANI, R., CHOI, H., AND BARANIUK, R. 2004. Forward: Fourier-wavelet regularized deconvolution for ill-conditioned systems. IEEE Trans. on Signal Processing 52 (Feburary), 418–433.
6. GULL, S. 1998. Bayesian inductive inference and maximum entropy. In Maximum Entropy and Bayesian Methods, J. Skilling, Ed. Kluwer, 54–71.
7. RICHARDSON, W. 1972. Bayesian-based iterative method of image restoration. Journal of the Optical Society of America A 62, 55–59.
8. TSUMURAYA, F., MIURA, N., AND BABA, N. 1994. Iterative blind deconvolution method using Lucy’s algorithm. Astron. Astrophys. 282, 2 (Feb), 699–708.
9. ZAROWIN, C. 1994. Robust, noniterative, and computationally efficient modification of van Cittert deconvolution optical figuring. Journal of the Optical Society of America A 11, 10 (October), 2571–83.
10. BASCLE, B., BLAKE, A., AND ZISSERMAN, A. 1996. Motion Deblurring and Superresolution from an Image Sequence. In ECCV (2), 573–582.
11. RAV-ACHA, A., AND PELEG, S. 2005. Two motion-blurred images are better than one. Pattern Recognition Letters, 311–317.
12. CANON INC., 2006. What is optical image stabilizer?

http://www.canon.com/ bctv/faq/optis.html.

13. LIU, X., AND GAMAL, A. 2001. Simultaneous image formation and motion blur restoration via multiple capture. In Proc. Int. Conf. Acoustics, Speech, Signal Processing, vol. 3, 1841–1844

14. BEN-EZRA, M., AND NAYAR, S. K. 2004. Motion-Based Motion Deblurring. IEEE Trans. on Pattern Analysis and Machine Intelligence 26, 6, 689–698.

15. ROTH, S., AND BLACK, M. J. 2005. Fields of Experts: A Framework for Learning Image Priors. In CVPR, vol. 2, 860–867.

16. TAPPEN, M. F., RUSSELL, B. C., AND FREEMAN, W. T. 2003. Exploiting the sparse derivative prior for super-resolution and image demosaicing. In SCTV.

17. WEISS, Y. 2001. Deriving intrinsic images from image sequences. In ICCV, 68–75.

18. APOSTOLOFF, N., AND FITZGIBBON, A. 2005. Bayesian video matting using learnt image priors. In Conf. on Computer Vision and Pattern Recognition, 407–414.

19. LEVIN, A., ZOMET, A., AND WEISS, Y. 2003. Learning How to Inpaint from Global Image Statistics. In ICCV, 305–312.

20. LEVIN, A., AND WEISS, Y. 2004. User Assisted Separation of Reflections from a Single Image Using a Sparsity Prior. In ICCV, vol. 1, 602–613.

21. FIELD, D. 1994. What is the goal of sensory coding? Neural Computation 6, 559–601.

22. SIMONCELLI, E. P. 2005. Statistical modeling of photographic images. In Handbook of Image and Video Processing, A. Bovik, Ed. ch. 4.

23. JORDAN, M., GHAHRAMANI, Z., JAAKKOLA, T., AND SAUL, L. 1999. An introduction to variational methods for graphical models. In Machine Learning, vol. 37, 183–233.

24. GEMAN, D., AND REYNOLDS, G. 1992. Constrained restoration and the recovery of discontinuities. IEEE Trans. on Pattern Analysis and Machine

Intelligence 14, 3, 367–383.

25. LUCY, L. 1974. Bayesian-based iterative method of image restoration. Journal. of Astronomy 79, 745–754.

26. BIGGS, D., AND ANDREWS, M. 1997. Acceleration of iterative image restoration algorithms. Applied Optics 36, 8, 1766–1775.

27. JANSSON, P. A. 1997. Deconvolution of Images and Spectra. Academic Press.

28. Introduction to Java Development. – Режим доступу: http://docs.opencv.org/2.4.4beta/doc/tutorials/introduction/desktop_java/java_dev_intro.html. – Дата доступу: 12.02.2015. – Назва з екрану.

29. Performance of BoofCV. – Режим доступу: http://boofcv.org/index.php?title=Performance:BoofCV. – Дата доступу: 29.02.2015. – Назва з екрану.

30. IPOL Journal – Image Processing On Line. – Режим доступу: http://www.ipol.im. – Дата доступу: 16.03.2015. – Назва з екрану.

31. Брюс Эккель. – Философия Java. Библитотека программиста. 4-е изд. / Эккель Б. – СПб.: Питер, 2009. – 640 с.: ил. ISBN 978-5-388-00003-3.

32. Кей Хорстман, Гари Корнел. – Java. Библиотека проффесионала, том 1. Основы / Хорстман, Кей С., Корнелл, Гари. – Пер. С англ. – М.: ООО “ИД Вильямс”, 2014. – 864 с.: ил. ISBN 978-5-8459-1869-7.

33. Джошуа Блох. – Java Эффективное программирование. 2-е изд. / Блох Дж. – “Лори”, 2014. – 461 ст. ISBN 978-5-85582-348-6.

34. Ильдар Хабибуллин. – Java 7. / Хабибуллин И. Ш. – СПб.: БХВ-Петербург, 2012. – 768 с.: ил. ISBN 978-5-9775-0735-6.

35. Рафаэль Гонсалес – Цифровая обработка изображений. / Гонсалес Р., Вудс Р. – Москва:Техносфера, 2005. – 1072. ISBN 5-94836-0258-8.

ДОДАТОК А

Лістинг Main.java

```
package com.mishailchyshyn;

import javax.swing.*;

public class Main {

     public static String APPLICATION_NAME = "wienerDeblur";

    public static void main(String[] args) {
         SwingUtilities.invokeLater(new Runnable() {
              public void run() {
                   showGUI();
              }
         });

    }

     private static void showGUI() {
          JFrame frame = new JFrame(APPLICATION_NAME);

     frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE
);
          frame.setSize(100, 100);

          frame.add(new DeblurGUI());

          frame.pack();
          frame.setVisible(true);
     }

}
```

Лістинг DeblurGUI.java

```
package com.mishailchyshyn;

import boofcv.abst.transform.fft.DiscreteFourierTransform;
import boofcv.alg.transform.fft.DiscreteFourierTransformOps;
import boofcv.struct.image.ImageFloat64;
import boofcv.struct.image.InterleavedF64;

import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
```

```
/**
 * Created by mishailchyshyn on 10.06.17.
 */
public class DeblurGUI extends JPanel implements ChangeListener
{

     public static final int WIDTH  = 1024;
     public static final int HEIGHT = WIDTH / 16 * 10;

     public static final int MIN_BLUR_RADIUS = 1;
     public static final int MAX_BLUR_RADIUS = 40;

     public static final int MIN_NOISE_LEVEL = 1;
     public static final int MAX_NOISE_LEVEL = 100;

     private BufferedImage input;
     private BufferedImage output;
     private BufferedImage kernel;

     private JLabel inputPanel;
     private JLabel outputPanel;
     private JLabel kernelPanel;

     private double radius;
     private double noise;

     private JSpinner radiusSpinner;
     private JSpinner noiseSpinner;

     private JButton open;
     private JButton save;

     public DeblurGUI() {

          try {
               input = ImageIO.read(new
File("/home/misha/Downloads/test.png"));
         } catch (IOException e) {
               System.out.println("Invalid input path");
         }

          radius = 9;
          noise = 2;

          open = new JButton();
          save = new JButton();

          initControls();
     }

     private void initControls() {
          radiusSpinner = new JSpinner(new SpinnerNumberModel(
```

```
                                        new Double(radius),
new Double(MIN_BLUR_RADIUS), new Double(MAX_BLUR_RADIUS), new
Double(0.1)));
          noiseSpinner = new JSpinner(new SpinnerNumberModel(
                                        new Double(noise),
new Double(MIN_NOISE_LEVEL), new Double(MAX_NOISE_LEVEL), new
Double(1.0)));

          radiusSpinner.addChangeListener(this);
          noiseSpinner.addChangeListener(this);

          inputPanel  = new JLabel(new ImageIcon());
          outputPanel = new JLabel(new ImageIcon());
          kernelPanel = new JLabel(new ImageIcon());

          open.setText("Open");
          save.setText("Save");

          doDeblurLayout();
     }

     private void doDeblurLayout() {
          setLayout(new GridBagLayout());

          GridBagConstraints c = new GridBagConstraints();

          c.fill = GridBagConstraints.BOTH;
          c.gridx = 0;
          c.gridy = 0;
          c.weightx = 0.1;
          c.weighty = 0.05;
          add(open, c);

          c.fill = GridBagConstraints.BOTH;
          c.gridx = 1;
          c.gridy = 0;
          c.weightx = 0.1;
          c.weighty = 0.05;
          add(save, c);

          c.fill = GridBagConstraints.BOTH;
          c.gridx = 0;
          c.gridy = 1;
          c.weightx = 0.15;
          c.weighty = 0.05;
          add(radiusSpinner, c);

          c.fill = GridBagConstraints.BOTH;
          c.gridx = 1;
          c.gridy = 1;
          c.weightx = 0.15;
          c.weighty = 0.05;
          add(noiseSpinner, c);
```

```
			c.fill = GridBagConstraints.BOTH;
			c.gridx = 0;
			c.gridy = 3;
			c.weightx = 0.3;
			c.weighty = 0.7;
			c.gridwidth = 2;
			add(inputPanel, c);

			c.fill = GridBagConstraints.BOTH;
			c.gridx = 0;
			c.gridy = 4;
			c.weightx = 0.3;
			c.weighty = 0.3;
			c.gridwidth = 2;
			add(kernelPanel, c);

			c.fill = GridBagConstraints.BOTH;
			c.gridx = 2;
			c.gridy = 0;
			c.weightx = 0.7;
			c.gridwidth = 1;
			c.gridheight = 4;
			add(outputPanel, c);
		}

		public Dimension getPreferredSize() {
			return new Dimension(WIDTH, HEIGHT);
		}

		@Override
		public void stateChanged(ChangeEvent e) {
			if (e.getSource() == noiseSpinner) {
				SpinnerNumberModel model =
(SpinnerNumberModel)noiseSpinner.getModel();
				noise = model.getNumber().doubleValue();
			}
			if (e.getSource() == radiusSpinner) {
				SpinnerNumberModel model =
(SpinnerNumberModel)radiusSpinner.getModel();
				radius = model.getNumber().doubleValue();
			}

			doDeblur();
		}

		private void doDeblur() {
			FocusBlur blur = new FocusBlur(radius);
			kernel = blur.getSquareKernel();

			DiscreteFourierTransform<ImageFloat64, InterleavedF64>
dft = DiscreteFourierTransformOps.createTransformF64();
```

```
            DeconvolutionTool deconvolutionTool = new
DeconvolutionTool(input,

blur.getKernelWithSize(input.getWidth(), input.getHeight()),
                                                     noise);

            long begin  = System.currentTimeMillis();
            output = deconvolutionTool.inverseDeblur();
            long end = System.currentTimeMillis();
            long runningTime = end - begin;
            System.out.println("doWienerDeblur running time (ms): "
+ runningTime);

            inputPanel.setIcon(new ImageIcon(input));
            kernelPanel.setIcon(new ImageIcon(kernel));
            outputPanel.setIcon(new ImageIcon(output));
      }
}
```

Лістинг FocusBlur.java

```
package com.mishailchyshyn;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

/**
 * Created by mishailchyshyn on 08.06.17.
 */
public class FocusBlur {

      private double radius;

      FocusBlur(double radius) {
            this.radius = radius;
      }

      public BufferedImage getKernelWithSize(int width, int
height) {
            BufferedImage kernelWH = new BufferedImage(width,
height, BufferedImage.TYPE_INT_ARGB);
            BufferedImage kernel = generateKernel();

            int xCenterWH = kernelWH.getWidth() / 2;
            int yCenterWH = kernelWH.getHeight() / 2;

            Graphics2D g2d = kernelWH.createGraphics();
```

```
            g2d.setColor(Color.black);
            g2d.fill(new Rectangle2D.Double(0.0, 0.0,
kernelWH.getWidth(), kernelWH.getHeight()));


      g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.S
RC_OVER, 1.0f));
            g2d.drawImage(kernel, xCenterWH - kernel.getWidth() /
2, yCenterWH - kernel.getHeight() / 2, null);
            g2d.dispose();

            return kernelWH;
      }

      public BufferedImage getSquareKernel() {
            return getKernelWithSize(100, 100);
      }

      private BufferedImage generateKernel() {
            int size = (int)(2 * radius + 6);
            size += size % 2;

            BufferedImage kernel = new BufferedImage(size, size,
BufferedImage.TYPE_INT_ARGB);

            Graphics2D g2d = kernel.createGraphics();

            g2d.setColor(Color.black);
            g2d.fill(new Rectangle2D.Double(0.0, 0.0,
kernel.getWidth(), kernel.getHeight()));

            g2d.setColor(Color.white);
            g2d.fill(new Ellipse2D.Double(3.0, 3.0, 2 * radius, 2 *
radius));
            g2d.dispose();

            return kernel;
      }

}
```

Лістинг DeconvolutionTool.java

```
package com.mishailchyshyn;

import boofcv.abst.transform.fft.DiscreteFourierTransform;
import boofcv.alg.misc.PixelMath;
import boofcv.alg.tracker.circulant.CirculantTracker;
import boofcv.alg.transform.fft.DiscreteFourierTransformOps;
import boofcv.core.image.ConvertBufferedImage;

import boofcv.core.image.ConvertImage;
import boofcv.gui.image.ImageGridPanel;
```

```
import boofcv.gui.image.ShowImages;
import boofcv.gui.image.VisualizeImageData;
import boofcv.struct.image.ImageFloat32;
import boofcv.struct.image.ImageFloat64;
import boofcv.struct.image.InterleavedF64;

import java.awt.image.BufferedImage;


/**
 * Created by mishailchyshyn .
 */
public class DeconvolutionTool {

     private DiscreteFourierTransform<ImageFloat64,
InterleavedF64> dft;

     private BufferedImage input;
     private BufferedImage kernel;

     private double K;
     private InterleavedF64 H;

     public DeconvolutionTool(BufferedImage inputImage,
BufferedImage kernel, double noiseLevel) {
          this.input  = inputImage;
          this.kernel = kernel;

          this.K = Math.pow(1.07, noiseLevel) / 10e-4;
          this.H = new InterleavedF64(kernel.getWidth(),
kernel.getHeight(), 2);

          dft = DiscreteFourierTransformOps.createTransformF64();
     }

     public BufferedImage doWienerDeblur() {
          BufferedImage result = new
BufferedImage(input.getWidth(), input.getHeight(),
BufferedImage.TYPE_INT_RGB);

          InterleavedF64 H = new
InterleavedF64(kernel.getWidth(), kernel.getHeight(), 2);
          ImageFloat64 h = toSpatialDomain(kernel);
          dft.forward(h, H);

          ImageFloat64 wienerInpRed   = new
ImageFloat64(input.getWidth(), input.getHeight());
          ImageFloat64 wienerInpGreen = new
ImageFloat64(input.getWidth(), input.getHeight());
          ImageFloat64 wienerInpBlue  = new
ImageFloat64(input.getWidth(), input.getHeight());

          for (int i = 0; i < input.getWidth(); i++) {
```

```
                for(int j = 0; j < input.getHeight(); j++) {
                     int color = input.getRGB(i, j);
                     int red   = (color & 0xff0000) >> 16;
                     int green = (color & 0xff00) >> 8;
                     int blue  = color & 0xff;

                     wienerInpRed.set(i, j, red);
                     wienerInpGreen.set(i, j, green);
                     wienerInpBlue.set(i, j, blue);
                }
           }

           long begin  = System.currentTimeMillis();

           ImageFloat64 wienerOutRed   = wiener(wienerInpRed, H);
           ImageFloat64 wienerOutGreen = wiener(wienerInpGreen,
H);
           ImageFloat64 wienerOutBlue  = wiener(wienerInpBlue, H);

           long end = System.currentTimeMillis();
           long runningTime = (end - begin) / 3;
           System.out.println("wiener algorithm average running
time (ms): " + runningTime);

           for (int i = 0; i < input.getWidth(); i++) {
                for(int j = 0; j < input.getHeight(); j++) {
                     int red   = (int)wienerOutRed.get(i, j);
                     int green = (int)wienerOutGreen.get(i, j);
                     int blue  = (int)wienerOutBlue.get(i, j);

                     int addBright = 0;
                     int RGB = (red   + (red   > 128 ? addBright :
-addBright) << 16) |
                                 (green + (green > 128 ? addBright
: -addBright) << 8) |
                                  blue  + (blue  > 128 ? addBright
: -addBright);

                     result.setRGB(i, j, RGB);
                }
           }

           for (int i = 0; i < result.getWidth() / 2; i++) {
                for(int j = 0; j < result.getHeight(); j++) {
                     int tempRGB = result.getRGB(i, j);
                     result.setRGB(i, j,
result.getRGB(result.getWidth() / 2 + i - 1, j));
                     result.setRGB(result.getWidth() / 2 + i - 1,
j, tempRGB);
                }
           }

           for (int i = 0; i < result.getWidth(); i++) {
```

```
                for(int j = 0; j < result.getHeight() / 2; j++) {
                     int tempRGB = result.getRGB(i, j);
                     result.setRGB(i, j, result.getRGB(i,
result.getHeight() / 2 + j - 1));
                     result.setRGB(i, result.getHeight() / 2 + j -
1, tempRGB);
                }
           }

           return result;
     }

     public ImageFloat64 wiener(ImageFloat64 g, InterleavedF64 H)
{
           InterleavedF64 G = new InterleavedF64(input.getWidth(),
input.getHeight(), 2);
           InterleavedF64 F = new InterleavedF64(input.getWidth(),
input.getHeight(), 2);

           dft.forward(g, G);

           InterleavedF64 H22 = new
InterleavedF64(input.getWidth(), input.getHeight(), 2);
           CirculantTracker.elementMultConjB(H, H, H22);

           for (int i = 0; i < F.width; i++) {
                for(int j = 0; j < F.height; j++) {
                     double gg = G.getBand(i, j, 0);
                     double gi = G.getBand(i, j, 1);

                     double hh = H.getBand(i, j, 0);
                     double hi = H.getBand(i, j, 1);

                     double h22  = H22.getBand(i, j, 0);
                     double h22i = H22.getBand(i, j, 1);

                     double mult1Re = divideComplex(1, 0.0, hh,
hi)[0];
                     double mult1Im = divideComplex(1, 0.0, hh,
hi)[1];

                     double mult2Re = divideComplex(h22, h22i, h22
+ K, h22)[0];
                     double mult2Im = divideComplex(h22, h22i, h22
+ K, h22)[1];

                     double multRe = multiplyComplex(mult1Re,
mult1Im, mult2Re, mult2Im)[0];
                     double multIm = multiplyComplex(mult1Re,
mult1Im, mult2Re, mult2Im)[1];

                     F.setBand(i, j, 0, multiplyComplex(multRe,
multIm, gg, gi)[0]);
```

```
                    F.setBand(i, j, 1, multiplyComplex(multRe,
multIm, gg, gi)[1]);
               }
          }

          return toSpatialDomain(F);
     }

     private double[] divideComplex(double a, double b, double c,
double d) {
          double[] result = new double[2];

          result[0] = (a * c + b * d) / (c * c + d * d);
          result[1] = (b * c - a * d) / (c * c + d * d);

          return result;
     }

     private double[] multiplyComplex(double a, double b, double
c, double d) {
          double[] result = new double[2];

          result[0] = a * c - b * d;
          result[1] = b * c + a * d;

          return result;
     }


     public ImageFloat64 toSpatialDomain(BufferedImage
inputImage) {
          ImageFloat32 f = new
ImageFloat32(inputImage.getWidth(), inputImage.getHeight());
          ConvertBufferedImage.convertFrom(inputImage, f);

          PixelMath.divide(f, 255.0f, f);

          ImageFloat64 ff = new
ImageFloat64(inputImage.getWidth(), inputImage.getHeight());
          ConvertImage.convert(f, ff);

          return ff;
     }

     private ImageFloat64 toSpatialDomain(InterleavedF64
inputImage) {
          ImageFloat64 f = new ImageFloat64(inputImage.width,
inputImage.height);

          dft.inverse(inputImage, f);
          PixelMath.multiply(f, 255.0f, f);

          return f;
```

```
	}

	private BufferedImage toBufferedImage(ImageFloat64 input) {
		ImageFloat32 f = new ImageFloat32(input.getWidth(),
input.getHeight());
		ConvertImage.convert(input, f);

		PixelMath.multiply(f, 255.0f, f);

		for (int i = 0; i < f.width; i++) {
			for(int j = 0; j < f.height; j++) {
				f.set(i, j, f.get(i, j) + 10);
			}
		}

		return ConvertBufferedImage.convertTo(f, null, true);
	}

	// Helpers

	public BufferedImage inverseDeblur() {
		InterleavedF64 G = new InterleavedF64(input.getWidth(),
input.getHeight(), 2);
		InterleavedF64 H = new
InterleavedF64(kernel.getWidth(), kernel.getHeight(), 2);
		InterleavedF64 F = new
InterleavedF64(kernel.getWidth(), kernel.getHeight(), 2);

		ImageFloat64 g = toSpatialDomain(input);
		ImageFloat64 h = toSpatialDomain(kernel);

		dft.forward(g, G);
		dft.forward(h, H);
		for (int i = 0; i < F.width; i++) {
			for(int j = 0; j < F.height; j++) {
				double G0 = G.getBand(i, j, 0);
				double G1 = G.getBand(i, j, 1);

				double H0 = H.getBand(i, j, 0);
				double H1 = H.getBand(i, j, 1);

				double F0 = (G0 * H0 + G1 * H1) / (H0 * H0 +
H1 * H1);
				double F1 = (H0 * G1 + G0 * H1) / (H0 * H0 +
H1 * H1);

				F.setBand(i, j, 0, F0);
				F.setBand(i, j, 1, F1);
			}
		}

//		showFrequency(F);
```

```
		return toBufferedImage(toSpatialDomain(F));
	}

	public BufferedImage convolution() {
		InterleavedF64 G = new InterleavedF64(input.getWidth(),
input.getHeight(), 2);
		InterleavedF64 H = new
InterleavedF64(kernel.getWidth(), kernel.getHeight(), 2);
		InterleavedF64 F = new
InterleavedF64(kernel.getWidth(), kernel.getHeight(), 2);

		ImageFloat64 g = toSpatialDomain(input);
		ImageFloat64 h = toSpatialDomain(kernel);

		dft.forward(g, G);
		dft.forward(h, H);

		DiscreteFourierTransformOps.multiplyComplex(H, G, F);

		return toBufferedImage(toSpatialDomain(F));
	}

	public BufferedImage forwardAndBack() {
		InterleavedF64 G = new InterleavedF64(input.getWidth(),
input.getHeight(), 2);

		ImageFloat64 g = toSpatialDomain(input);

		dft.forward(g, G);

		return toBufferedImage(toSpatialDomain(G));
	}

	// Show in BufferedImage

	private void showMagnitudeAndPhase(InterleavedF64 F, String
name) {
		ImageFloat64 magnitude = new
ImageFloat64(F.width,F.height);
		ImageFloat64 phase = new
ImageFloat64(F.width,F.height);

		// Make a copy so that you don't modify the input
		F = F.clone();

		// shift the zero-frequency into the image center, as
is standard in image processing
		DiscreteFourierTransformOps.shiftZeroFrequency(F,true);

		// Compute the transform's magnitude and phase
		DiscreteFourierTransformOps.magnitude(F,magnitude);
		DiscreteFourierTransformOps.phase(F, phase);
```

```
            // Convert it to a log scale for visibility
            PixelMath.log(magnitude, magnitude);

            ImageFloat32 m = new ImageFloat32(F.width,F.height);
            ImageFloat32 p = new ImageFloat32(F.width,F.height);

            ConvertImage.convert(magnitude, m);
            ConvertImage.convert(phase, p);

            // Display the results
            BufferedImage visualMag =
VisualizeImageData.grayMagnitude(m, null, -1);
            BufferedImage visualPhase =
VisualizeImageData.colorizeSign(p, null, Math.PI);

            ImageGridPanel dual = new
ImageGridPanel(1,2,visualMag,visualPhase);
            ShowImages.showWindow(dual, "Magnitude and Phase of " +
name);
      }

      private void showFrequency(InterleavedF64 frequency) {
            System.out.println("Frequency:----------------");
            for (int i = 0; i < frequency.width; i++) {
                  System.out.println();
                  for(int j = 0; j < frequency.height; j++) {
                        System.out.print(frequency.getBand(i, j, 0) +
" " + frequency.getBand(i, j, 1) + "i ");
                  }
            }
      }

      // Show in text
      private void showSpatial(ImageFloat64 spatial) {
            System.out.println("Spatial64:----------------");
            for (int i = 0; i < spatial.width; i++) {
                  System.out.println();
                  for(int j = 0; j < spatial.height; j++) {
                        System.out.print(spatial.get(i, j) + " ");
                  }
            }
      }

      private void showSpatial(ImageFloat32 spatial) {
            System.out.println("Spatial32:----------------");
            for (int i = 0; i < spatial.width; i++) {
                  System.out.println();
                  for(int j = 0; j < spatial.height; j++) {
                        System.out.print(spatial.get(i, j) + " ");
                  }
            }
      }
}
```