КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

На правах рукопису

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від _____ 2024 року, протокол № ___.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
на тему:
«Використання бібліотеки OpenCv для обробки гартманограм»

Виконав:
студент 4-го курсу
денної форми навчання
спеціальності 172 - Телекомунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»
Ясінський Ярослав Валерійович

Науковий керівник:
канд. фіз.-мат. наук, асистент
КОТОВ МИХАЙЛО МИКОЛАЙОВИЧ_____

Рецензент:
канд. фіз.-мат. наук, доцент
Оберемок Євген Анатолійович _____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент_____ Ясінський Ярослав Валерійович

Київ 2024

РЕФЕРАТ

Дипломна робота: 53 с., 2 табл., 16 рис., 1 дод. (13с.), 6 джерел.

OpenCV, СЕНСОР ШЕКА-ГАРТМАНА, ОБРОБКА ЗОБРАЖЕНЬ

Об’єкт розроблення – програма обробки гартманограм з використанням бібліотеки комп'ютерного зору OpenCv з інтерфейсом користувача.

Мета роботи – розробка програмного забезпечення для автоматичного аналізу зображень з використанням бібліотеки OpenCV. Програма повинна надавати зручний інтерфейс для завантаження двох зображень, відображати їх для візуальної перевірки та виконувати попередню обробку зображень, таку як згладжування та підвищення контрастності, для полегшення подальшого аналізу. Основна мета обробки – знайти центри мас точок на кожному зображенні та порівняти знайдені координати точок на двох зображеннях, обчислюючи різницю координат (дельта X та дельта Y) між відповідними точками.

Програма відображає результати аналізу у вигляді таблиці в інтерфейсі користувача, надаючи можливість збереження результатів в файл для подальшого використання. Інтерфейс програми забезпечує зручність і зрозумілість взаємодії користувача з програмою. Це дозволяє користувачам легко завантажувати зображення, переглядати їх, обробляти та отримувати результати аналізу.

Для реалізації поставлених завдань використовується бібліотека OpenCV, яка є потужним інструментом для роботи з зображеннями та відео. OpenCV використовується для зчитування та відображення зображень, застосування алгоритмів попередньої обробки, знаходження центрів мас точок на зображеннях та обчислення координат та їх різниці між двома зображеннями.

Програма реалізована за допомогою мови програмування C++ з використанням бібліотеки OpenCV та фреймворку Qt для створення графічного інтерфейсу користувача.

ЗМІСТ

ВСТУП

Сенсор Шека–Гартмана є ефективним інструментом для аналізу хвильових фронтів та визначення їх аберацій, дозволяючи отримувати високоточні дані про розподіл світла через оптичні системи. Він складається з масиву мікролінз, де кожна фокусує частину світла на детектор, створюючи візерунок з точок світла.

Коли світло проходить через масив мікролінз, воно розбивається на малі промені, які фокусуються на детекторі. Аналізуючи положення цих точок, можна отримати інформацію про деформації хвильового фронту. В ідеальних умовах точки фокусуються в передбачених місцях, але при наявності аберацій вони зміщуються, що дозволяє виміряти нахили хвильового фронту і відновити його загальну форму.

Ця дипломна робота спрямована на розробку алгоритму для пошуку центрів мас у масиві точок, що є результатом роботи сенсора Шека–Гартмана. Точність визначення центрів мас є критично важливою для коректного відновлення хвильового фронту та виявлення аберацій.

У майбутньому запропонований алгоритм може бути використаний у різних галузях, де потрібна висока точність оптичних вимірювань, таких як астрономія, офтальмологія та адаптивна оптика. Це допоможе коригувати зображення з телескопів, діагностувати зорові аберації та покращувати якість зображень у наукових та технічних застосуваннях.

Метою роботи є розробка та тестування алгоритму для автоматизації пошуку центрів мас точок сенсора Шека–Гартмана, що підвищить точність і ефективність аналізу хвильових фронтів у різних оптичних системах.

РОЗДІЛ 1. МАТРИЦЯ МІКРОЛІНЗ. ОСНОВНІ ВІДОМОСТІ ПРО ДАТЧИК ШЕКА-ХАРТМАНА. АЛГОРИТМИ АНАЛІЗУ ТА ЗЧИТУВАННЯ ЗОБРАЖЕНЬ.

Основним заданням мого диплому є аналіз зображення яке виникає в результаті дії матриці мікро лінз та датчик Шака - Хартмана в цьому розділі буде детально розписаний механізм їх роботи, застосування та використання. Також буде наведений перелік основних методів аналізу та зчитування зображень, а також їх аналіз для подальшої розробки практичного алгоритму для данної задачі.

1.1 Датчик Шека–Гартмана та постановка задачі

Датчик хвильового фронту Шека–Гартмана (або іноді Гартмана–Шака) є найпоширенішим типом датчика хвильового фронту, названого на честь Йоганнеса Франца Гартмана та Роланда Шака. Його можна використовувати для вимірювання форми хвильового фронту падаючого світла, наприклад, від ослабленого лазерного променя або світла зірки в оптичному телескопі . Зокрема, він часто служить важливим компонентом систем адаптивної оптики[1].

Основною складовою такого датчика є матриця мікролінз.

Мікролінзові матриці представляють собою масиви невеликих лінз(рисунок 1.1), які зазвичай формують періодичний візерунок квадратного або шестикутного типу з кроком кілька сотень або десятків мікрометрів[2]. Вони доступні як окремі оптичні компоненти, часто оснащені антибліковим покриттям з обох сторін, і можуть бути частиною більшої збірки, наприклад, у датчиках зображення CCD і CMOS для підвищення ефективності збирання світла.

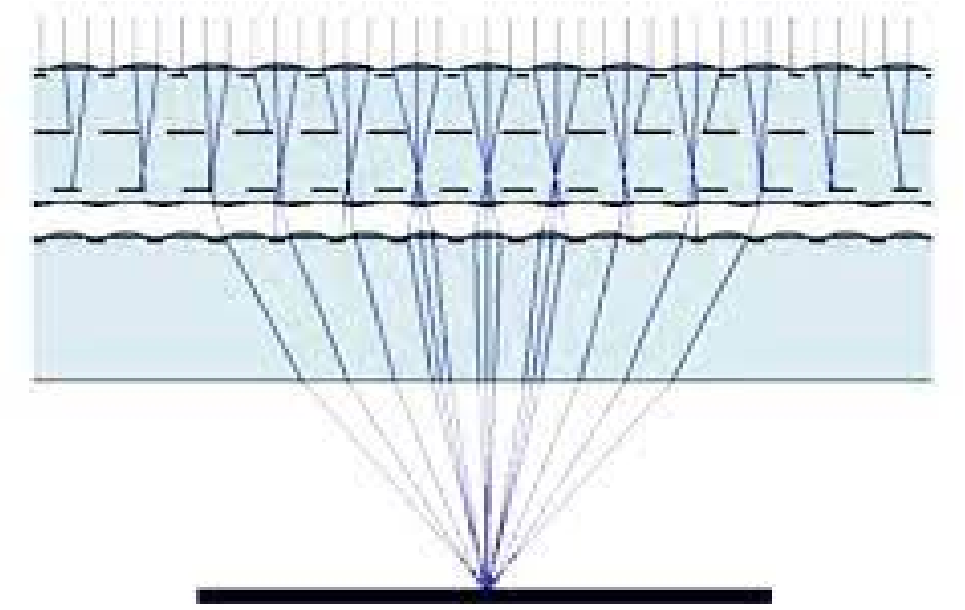


Рисунок 1.1 Схематичне зображення роботи масиву мікролінз

Методи виготовлення мікролінз поділяються на прямі та непрямі. Прямі методи, такі як гаряче тиснення, струменевий друк, електрогідродинамічний струменевий друк, термічне оплавлення фоторезисту, лазерні методи та самоскладання, не потребують виготовлення маски. Непрямі методи включають вологе травлення і м'яку літографію. Кожен з цих методів має свої переваги та недоліки, що впливає на якість і розмір мікролінз[2].

Мікролінзи є ключовим елементом мікрооптико-електромеханічних систем завдяки своїй мініатюрності, високій інтегрованості та великому полю зору. Вони широко використовуються у створенні зображень, підвищенні ефективності світловипромінювальних діодів та в різних галузях, включаючи цивільні, військові, аерокосмічну та біомедичну сфери.

1.1.1 Принцип дії

Оптична установка датчика складається з масиву мікролінз і датчика зображення , який встановлено у фокальній площині масиву мікролінз (його можна назвати масивом фокальної площини). Оригінальний винахід Йоганнеса Франца Гартмана використовував маску Гартмана, яка являє собою ряд отворів замість лінз; Наприкінці 1960-х років Роланд Шек і Бен Платт запровадили використання матриць лінз, що значно підвищило чутливість цих пристроїв і,

таким чином, дозволило вимірювати хвильовий фронт на значно нижчих рівнях інтенсивності, що важливо, наприклад, для застосувань в астрономії. Частково ще використовуються хвильові датчики з маскою Хартмана (з отворами); вони називаються датчиками хвильового фронту Гартмана .

Принцип роботи такого датчика хвильового фронту досить простий. Кожна лінза пристрою фокусує вхідне випромінювання в точку на датчику (Рисунок 1.2), і положення цієї точки вказує на орієнтацію хвильових фронтів , усереднену по площі входу лінзи[1].

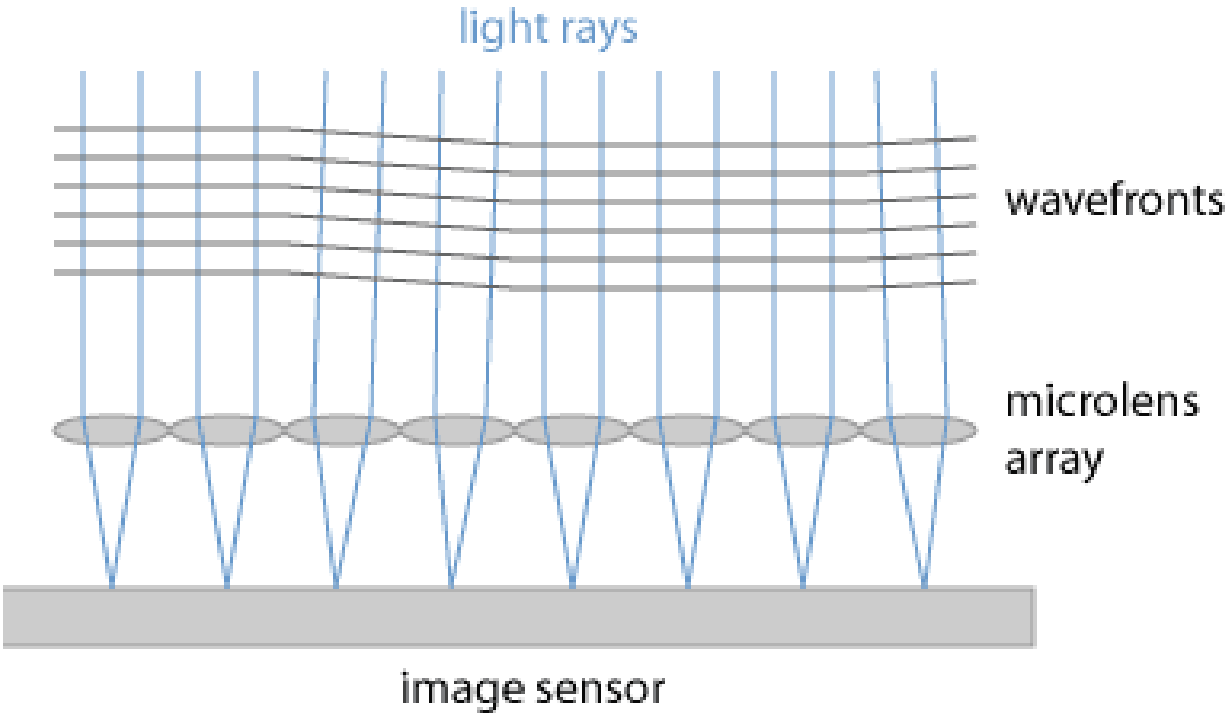


Рисунок 1.2 Оптична установка датчика хвильового фронту Шека–Гартмана, на якому показано один ряд мікролінз (лінз) над датчиком зображення.

В результаті дії сенсора буде отриманий масив плям на площині, при цьому можливі викривлення. В подальшому деякий комп’ютера використовується для обчислення положення плям позаду всіх лінз (сегментів лінзи) на основі отриманого зображення та оцінки спотворень хвильового фронту по всій зоні входу датчика на основі цих даних. Створення програми для обчислення положення цих плям і буде моїм завданням.

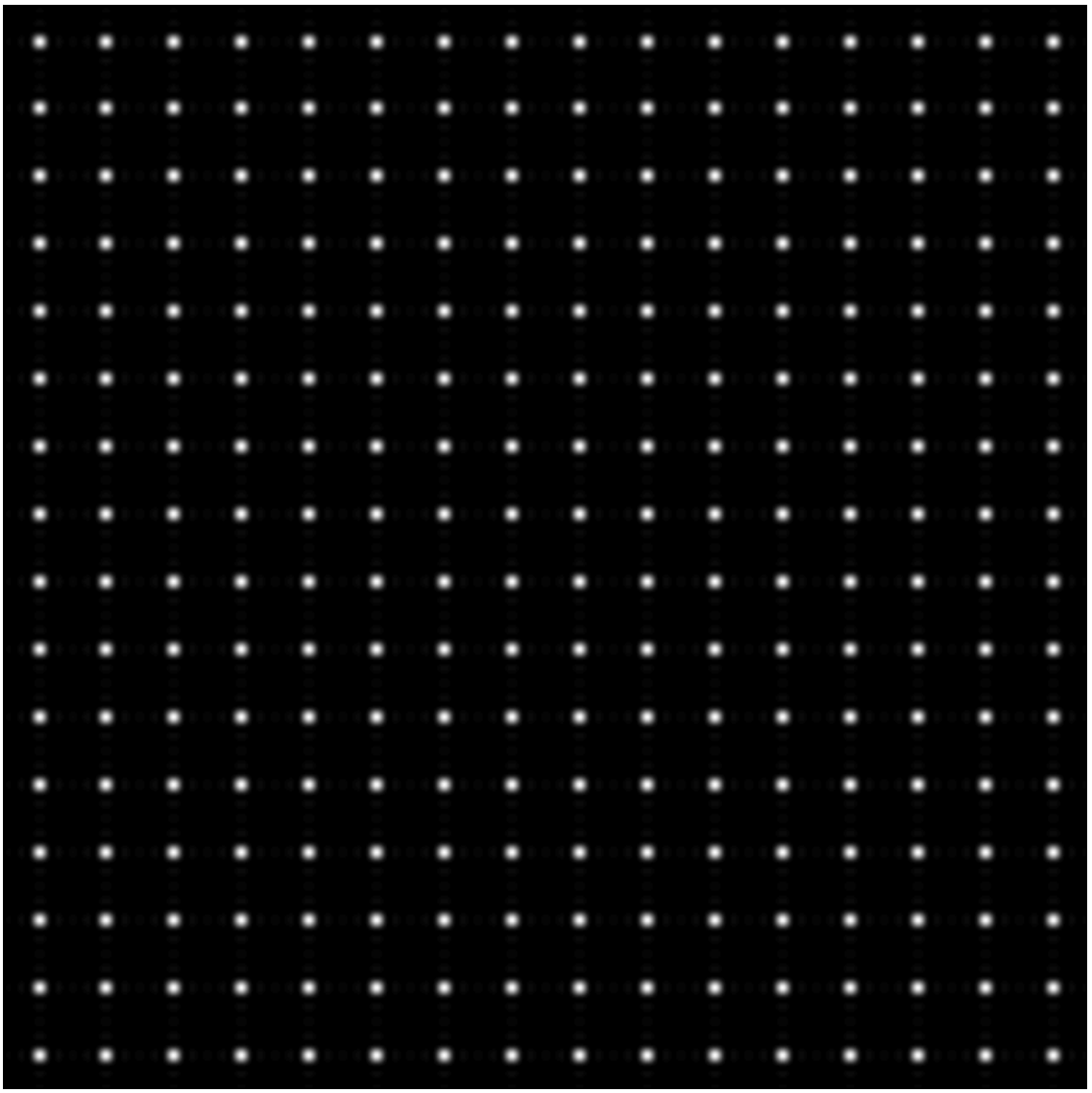


Рисунок 1.3 Змодельований приклад

На Рисунку 1.3 представлений звичайний результат роботи датчика.

Технічні характеристики

Найважливіші специфікації датчиків хвильового фронту стосуються таких аспектів:

- Область вимірювання (діафрагма): зазвичай це прямокутна область, ширина та висота якої вказуються. Як правило, вони мають розміри від 5 мм до 20 мм, але набагато більші датчики виготовляються, наприклад, для астрономічних застосувань.
- Просторова роздільна здатність: визначається кроком лінз (а не кроком пікселів датчика). Як правило, це кілька сотень мікрометрів. Тому

зазвичай необхідно використовувати якийсь розширювач променя для типового лазерного променя . Можна мати лише, наприклад, 80 × 50 сегментів детектора, навіть якщо датчик зображення має більше 1500 × 1000 пікселів. Для вимірювання коефіцієнтів Церніке вищого порядку потрібна більша кількість сегментів детектора.

- Кутовий діапазон: він по суті обмежений кроком лінз, поділеним на фокусну відстань масиву.
- Динамічний діапазон щодо оптичної фази : він може бути набагато більшим, ніж 2π і набагато більше, ніж у багатьох інтерферометрів . Зверніть увагу, що прилад вимірює не абсолютні значення фази, а похідні фази по поперечних координатах. За допомогою інтегрування можна отримати досить великі фазові екскурсії, якщо вони не надто швидкі.
- Динамічний діапазон щодо оптичної вхідної потужності: зауважте, що датчик ніколи не повинен бути насиченим, але також не повинен працювати із занадто низькою оптичною вхідною потужністю. (В ідеалі прилад показує під час роботи, чи є пікові інтенсивності безперечним діапазоном.) На практиці, можливо, доведеться використовувати змінний оптичний аттенюатор для регулювання оптичної вхідної потужності – але, звичайно, без внесення істотних додаткових спотворень хвильового фронту. Вплив навколишнього освітлення може ще більше зменшити ефективний доступний динамічний діапазон.
- Точність хвильового фронту часто визначається як частка оптичної довжини хвилі . Може бути λ/10 для простих пристроїв або порядку λ/100 для особливо точних датчиків. Зауважте, що точність хвильового фронту може бути не такою високою, як чутливість хвильового фронту: можна виявити невеликі зміни хвильового фронту без можливості точної кількісної оцінки значень. Крім того, найвища точність вимірювання може бути досягнута лише за допомогою деякого усереднення, яке може зменшити досягнуту швидкість вимірювання в термінах вимірювань за

секунду або (для ковзних середніх) просто уповільнити реакцію на зміни хвильового фронту.

- Швидкість вимірювання зазвичай вказується як кількість вимірювань за секунду; її також називають частотою кадрів і подають у одиницях кадрів за секунду (fps). Однак зауважте, що досягнута частота кадрів може бути значно нижчою за максимальну частоту кадрів використовуваного датчика зображення, оскільки обчислення можуть зайняти деякий час. Швидкі датчики хвильового фронту можуть забезпечувати тисячі форм хвильового фронту за секунду. Великі області вимірювання зазвичай поєднуються з нижчою частотою кадрів.
- Іншими цікавими практичними аспектами можуть бути геометричні розміри вимірювальної головки, варіанти монтажу, пристрій, що використовується для обробки даних (наприклад, внутрішня електроніка або програмне забезпечення ПК), гнучкість використовуваного програмного забезпечення, електронні та/або програмні інтерфейси та вимоги щодо процедур калібрування.
- Деякі пристрої дозволяють замінювати масив мікролінз, щоб можна було проводити вимірювання з різними кроками лінз. Для деяких вимірювань можна, наприклад, віддати перевагу масиву з низьким кроком для високої просторової роздільної здатності, навіть якщо це зменшує кутовий діапазон і, можливо, отриману точність хвильового фронту[1].

1.1.2 Застосування датчиків хвильового фронту Шека–Гартмана

Астрономічні телескопи

Датчики хвильового фронту Шека–Гартмана часто використовуються в контексті адаптивної оптики , зокрема для астрономічних телескопів . Вони необхідні для вимірювання орієнтації хвильового фронту, наприклад, світла від далекої зірки або лазерної зірки ; результати використовуються для корекції орієнтації хвильового фронту за допомогою деформівного дзеркала .

Характеристика оптики

Характеристика оптики може включати оцінку оптичних аберацій лінз або об'єктивів під час їх проходження світла. Наприклад, для перевірки лінзи без аберацій тестовий лазерний промінь проходить через неї за відповідних умов, а потім проходить до датчика хвильового фронту. Це дозволяє уникнути надмірних відхилень фази на датчику.

Інші методи працюють у відображенні, де лазерний промінь надсилається на дзеркало, і аналізуються хвильові фронти відбитого світла. Також можна використовувати високоякісну лінзу для створення плоских хвильових фронтів перед датчиком хвильового фронту.

Системи вимірювання хвильового фронту можуть включати джерело світла, різні оптичні компоненти, гнучке кріплення для перевірки компонентів і комп'ютерну систему для обробки даних та відображення результатів.

Офтальмологічна діагностика

Ще одне застосування в офтальмологічній діагностиці - вимірювання оптичних аберацій ока. Наприклад, за допомогою лазера ближнього інфрачервоного діапазону можна створити освітлену пляму на сітківці та проаналізувати орієнтацію хвильового фронту світла, що проходить через лінзу ока. Це дозволяє виміряти аберації, наприклад, ступінь астигматизму, для подальшої корекції за допомогою окулярів за рецептом. Також можна виміряти форму рогівки за допомогою відбитого світла.

Датчики хвильового фронту можна використовувати і для отримання образів ока, покращуючи якість зображення та просторову роздільну здатність шляхом корекції спотворень хвильового фронту за допомогою деформівного дзеркала або рідкокристалічного модулятора. Крім того, оптична когерентна томографія може поєднуватися з адаптивною оптикою для покращення бокової роздільної здатності.

Характеристика лазерного променя

Датчики хвильового фронту також можна використовувати для визначення характеристик лазерних променів . На відміну від пристроїв, які

вимірюють лише профілі інтенсивності, наприклад, щоб кількісно визначити якість променя за допомогою добутку параметрів променя або коефіцієнта M^2, датчики хвильового фронту можна використовувати для отримання повного профілю оптичної фази. Потім отримує більш повну інформацію, яка, можливо, допоможе визначити причину можливого погіршення якості променя.

Застосування отриманої інформації з датчика та постановка задачі

Розрахунок спот-позицій - це метод визначення точного положення об'єктів на зображенні, що використовується в області обробки зображень і комп'ютерного зору. Зазвичай він вимагає аналізу піксельної інформації з цифрових зображень. Однак через обмежену роздільну здатність датчика зображення необхідно враховувати, що положення об'єктів на зображенні не завжди відповідає простому вибору пікселя з найвищою оптичною інтенсивністю.

Тут важливим є уникнення простої прив'язки до пікселів і використання методу «центру ваги». Суть цього методу полягає в тому, щоб замість простого вибору пікселя з найбільшою інтенсивністю обчислити центр мас, або центр ваги, який враховує розподіл інтенсивності на зображенні. Це дає більш точну інформацію про положення об'єкту, особливо коли фокус променя впливає на декілька пікселів.

Крім того, використання більш складних обчислювальних алгоритмів дозволяє краще пригнічувати шум та отримувати ще більш точні дані. Це особливо важливо в ситуаціях з великими відхиленнями фази або значним впливом шуму на зображення.

Принцип роботи цього методу також можна пояснити через оптику Фур’є. Згідно з цією концепцією, кожна лінза утворює профіль інтенсивності у фокальній площині, що пов’язаний з просторовим перетворенням Фур’є комплексного амплітудного розподілу падаючого випромінювання. Це дозволяє отримати інформацію про структуру об'єктів на зображенні.

Розрахунок спотворень хвильового фронту можна здійснити шляхом обчислення локальної орієнтації хвильового фронту. Для цього достатньо розділити спотове зміщення на відстань між лінзами та поверхнею сенсора. Отримані орієнтації можуть бути використані для відновлення поля хвильового фронту та подальшого обчислення профілю амплітуди.

Далі можна використовувати оптимізовані алгоритми для мінімізації чутливості до шуму вимірювань. Це важливо для точного визначення профілю амплітуди та якості променя, наприклад, за допомогою коефіцієнта M^2 або коефіцієнта Штреля. Також можна застосувати розкладання виміряних оптичних аберацій за допомогою поліномів Церніке для отримання коефіцієнтів Церніке у реальному часі.

Велика фокусна відстань лінз є корисною для досягнення високої роздільної здатності для орієнтації хвильового фронту. Однак це може обмежити прийнятний кутовий діапазон для вхідного світла, оскільки збільшення фокусної відстані може призвести до занадто великого зсуву плями на сенсорі, спричиняючи перехресні перешкоди із сусідніми сегментами датчика. Тому можливо, було б краще використовувати меншу фокусну відстань та покращити роздільну здатність за допомогою датчика з меншою відстанню між пікселями.

Щодо можливих артефактів, слід зазначити, що перехресні перешкоди можуть виникнути при занадто великих кутах падіння світла на лінзу. Крім того, можуть виникнути проблеми з розсіюванням світла на краях лінз або з паразитними відблисками. Тому для отримання високоякісних результатів вимірювань необхідна гарна оптична якість матриці мікролінз, а також оптимізована обробка даних за допомогою відповідних програмних алгоритмів. Отже для успішної розробки алгоритма, враховуючи все вищеперечислене потрібно проаналізувати основні існуючі способи обробки зображень.

1.2 Перелік алгоритмів аналізу зображення в контексті конкретної задачі.

Зображення можна представити як двовимірну функцію F(x,y), де x і y – просторові координати. Амплітуда F при певному значенні x,y відома як інтенсивність зображення в цій точці. Якщо x, y і значення амплітуди кінцеве, ми називаємо це цифровим зображенням. Це масив пікселів, упорядкованих у стовпці та рядки. Пікселі — це елементи зображення, які містять інформацію про інтенсивність і колір. Зображення також можна представити в 3D, де x, y і z стають просторовими координатами. Пікселі розташовані у вигляді матриці. Це зображення називається RGB.

Існують різні види зображень:

- RGB-зображення: воно містить три шари 2D-зображення, ці шари червоного, зеленого та синього каналів.
- Зображення у відтінках сірого: ці зображення містять відтінки чорного та білого та містять лише один канал.

В нашому випадку буде саме другий варіант[3].

1.2.1 Обробка морфологічного зображення

Обробка морфологічного зображення - це метод, спрямований на виправлення недоліків бінарних зображень, що можуть виникати внаслідок наявності шуму. Для цього використовуються морфологічні операції, такі як розширення та ерозія, які також можуть використовуватися для згладжування зображень.

Морфологічні операції можуть бути застосовані не лише до бінарних зображень, але й до зображень в градаціях сірого. Вони є нелінійними операціями, що використовують інформацію про структуру ознак зображення. Ці операції аналізують зображення за допомогою структурного елементу, який розміщується в різних частинах зображення та порівнюється з сусідніми пікселями.

Розглянемо дві основні морфологічні операції: розширення та ерозію. Розширення полягає у додаванні пікселів до меж об'єкта на зображенні, тоді як ерозія видаляє пікселі з меж об'єкта. Кількість доданих або видалених пікселів залежить від розміру структурного елемента, який є невеликою матрицею, складеною з нулів та одиниць[3].

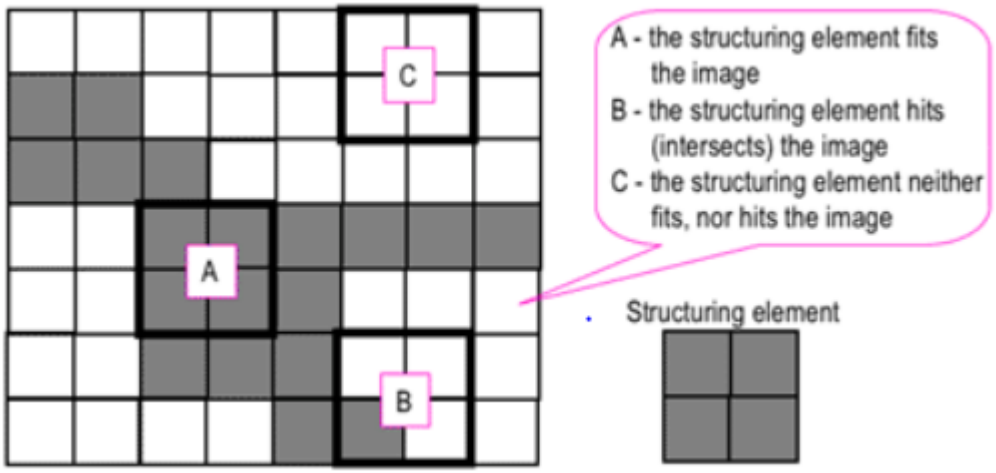


Рисунок 1.4 Принцип роботи морфологічної обробки

Квадратний структурний елемент «А» вписується в об’єкт, який ми хочемо вибрати, «В» перетинає об’єкт, а «С» знаходиться поза об’єктом(Рисунок 1.4*)*.

Шаблон нуль-один визначає конфігурацію структурного елемента. Це залежить від форми об’єкта, який ми хочемо вибрати. Центр структурного елемента ідентифікує піксель, що обробляється.

В ситуації з масивом точок це досить корисний метод обробки зображень, який можна застосувати щоб позбутись більшої частини шумів. Проблема цього методу в тому, що деякі точки можуть бути занадто темні і в результаті видалені. Це треба врахувати при розробці алгоритма.

Дилатація

Дилатація є однією з базових операцій морфологічної обробки зображень, що широко застосовується в комп'ютерному баченні та обробці цифрових зображень. Ця операція, яка належить до категорії нелінійних перетворень, дозволяє розширити об’єкти на зображенні, заповнити прогалини та з’єднати

близько розташовані компоненти. Основним принципом дилатації є розширення меж об’єкта шляхом додавання пікселів до його контурів відповідно до визначеного структурного елемента.

Структурний елемент (СЕ) є ключовим компонентом дилатації, який визначає форму і розмір розширення. Зазвичай СЕ є невеликою матрицею, що містить одиниці і нулі, які описують форму, яку слід використовувати для обробки зображення. Наприклад, квадратний або круглий СЕ часто використовується для розширення об'єктів на зображенні. Під час дилатації СЕ переміщається по всьому зображенню, і якщо хоча б один піксель з об'єкта співпадає з одиницею СЕ, центральний піксель зображення стає частиною об'єкта.

Основна мета дилатації в обробці зображень полягає в розширенні об'єктів та зменшенні впливу шумів. Ця операція є особливо корисною для поліпшення якості бінарних зображень, де вона може використовуватися для згладжування контурів, заповнення дрібних прогалин і з'єднання розірваних частин об'єкта. Завдяки цим властивостям, дилатація знаходить широке застосування в практичних задачах, таких як розпізнавання об'єктів, сегментація зображень, аналіз медичних зображень та автоматизація промислових процесів[3].

Важливим аспектом використання дилатації є вибір відповідного СЕ та параметрів обробки. Неправильний вибір СЕ може призвести до небажаних результатів, таких як надмірне розширення об'єктів або заповнення небажаних ділянок. Тому, для досягнення оптимальних результатів, необхідно ретельно налаштовувати параметри дилатації відповідно до специфіки оброблюваних зображень та поставлених задач.

Треба врахувати всі ці фактори при розробці програми[3].

1.2.2 Обробка зображень Гауса

Розмиття за Гаусом, відоме також як гаусівське згладжування, є методом розмиття зображення, який базується на використанні функції Гауса.

Використовується для приглушення шуму та зменшення рівня деталізації на зображенні. Візуально, ця техніка розмиття подібна до ефекту перегляду зображення через напівпрозорий екран. Вона знаходить своє застосування у комп'ютерному зорі для покращення зображення на різних масштабах або в якості методу збільшення даних у глибокому навчанні.

Основна функція Гауса може бути виражена наступним чином(1):

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad (1)$$

Де:

G(x) — це значення Гаусіана в точці xxx.

σ — це стандартне відхилення розподілу, яке визначає ширину Гаусіана. Чим більше значення σ, тим ширше і плавніше буде крива[4].

На практиці, для досягнення оптимальних результатів, використовують властивість відокремлюваності розмиття за Гаусом, розбивши процес на два проходи. Під час першого проходу застосовується одновимірне ядро для розмиття зображення лише у горизонтальному або вертикальному напрямку. Під час другого проходу те саме одновимірне ядро використовується для розмиття в залишкових напрямках. Ефект цього процесу аналогічний згортанню з двовимірним ядром за один прохід. Розглянемо приклад, щоб краще зрозуміти, як фільтри Гауса впливають на зображення. З прикладу видно, що данний метод ефективний в випадках коли деталі не важливі. Цей метод можна використати для розробки алгоритма.

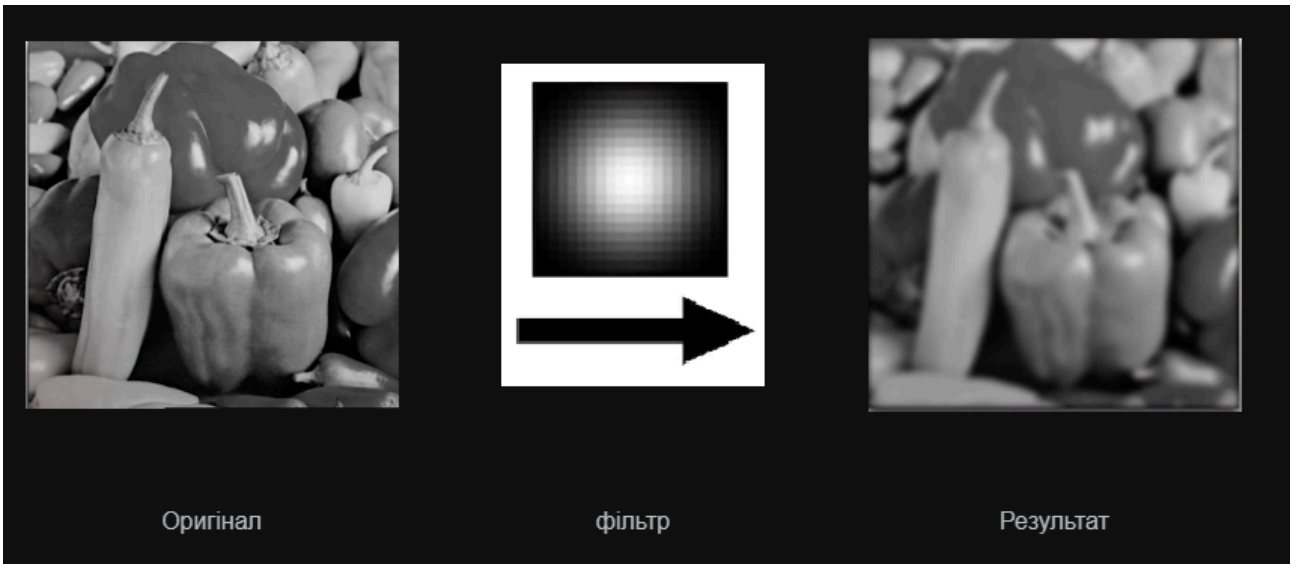


Рисунок 1.5. Робота фільтру.

На рисунку 1.5 бачите, що деякі краї мають трохи менше деталей. Фільтр надає більшої ваги пікселям у центрі, ніж пікселям, розташованим подалі від центру. Гауссові фільтри є фільтрами низьких частот, тобто послаблюють високі частоти. Він зазвичай використовується для виявлення країв

Виявлення країв при обробці зображень

Виявлення країв — це техніка обробки зображень для знаходження меж об’єктів на зображеннях. Він працює шляхом виявлення розривів яскравості.

Це може бути дуже корисним для отримання корисної інформації із зображення, оскільки більшість інформації про форму укладено по краях. Класичні методи визначення країв працюють шляхом виявлення розривів яскравості.

Він може швидко реагувати, якщо на зображенні буде виявлено шум під час виявлення варіацій рівнів сірого. Краї визначаються як локальні максимуми градієнта.

Найпоширенішим алгоритмом виявлення краю є алгоритм виявлення краю Sobel . Оператор виявлення Sobel складається зі згорткових ядер 3*3. Просте ядро Gx і повернуте на 90 градусів ядро Gy(2). Окремі вимірювання проводяться шляхом нанесення ядра окремо на зображення.

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * Image\ matrix$$

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * Image\ matrix \quad (2)$$

* позначає операцію згортки обробки двовимірного сигналу.

Результуючий градієнт можна розрахувати як(3):

$$G = sqrt(Gx^2 + Gy^2) \quad (3)$$

Де G представляє величину градієнта.

Gx та Gy представляють компоненти градієнта у напрямках осей x та y відповідно[4].

РОЗДІЛ 2. НАПИСАННЯ КОДУ ДЛЯ ВИРІШЕННЯ ТЕХНІЧНОЇ ЗАДАЧІ

2.1 Постановка основних завдань алгоритму

Після опису основних визначень та теоретичної частини моєї роботи можна переходити до написання коду. Для цієї програми я буду використовувати мову програмування C++. Ця мова гарно підходить для подібних програм завдяки великій кількості різноманітних бібліотек а також зручний функціонал мови.

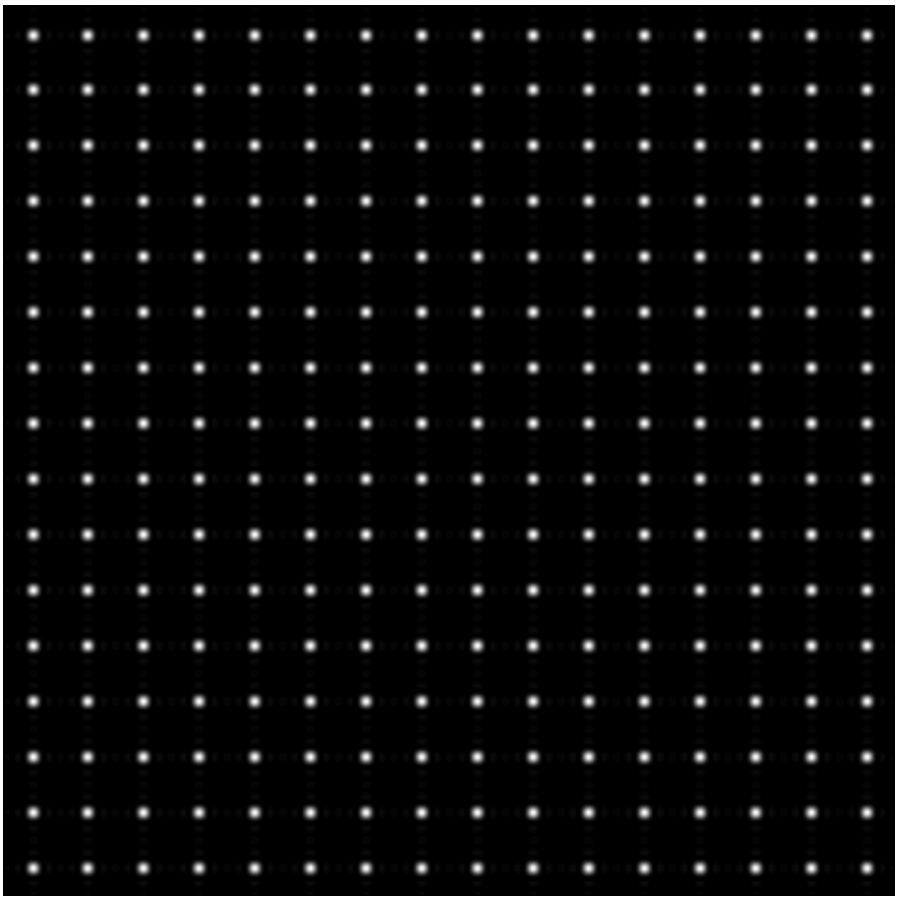


Рисунок 2.1. Приклад зображення

Алгоритм повинен виконати наступні завдання:

В програму завантажуються два зображення формату BMP, на зображеннях присутній масив світлих точок на темному фоні(як на рисунку 2.1), масив точок може бути викривлено. Завдання алгоритму прибрати можливі шуми які можуть заважати аналізувати зображення при цьому не втратити точки(для цього можна використати метод Гауса, чи інший алгоритм обробки зображень), визначити координати центра мас кожної точки на обох зображеннях, співставити

відповідні точки з обох зображень і в результаті вивести таблицю з координатами кожної точки обох зображень, а також відхилення відповідних точок зображення 2 від зображення 1. Кожну дію для зручності писання коду винису в окрему функцію.

2.2 Розроблення програми

Для написання програми скористаємося бібліотеками OpenCV для обробки зображень та Qt для створення інтерфейсу користувача.

Короткий опис задіяних бібліотек:

OpenCV (Open Source Computer Vision Library) є відкритою бібліотекою для комп'ютерного зору та машинного навчання, яка має інтерфейси для C++, Python, Java та підтримує різні операційні системи, включаючи Linux, macOS та Windows. Вона містить понад 2500 алгоритмів, які використовуються для різноманітних завдань, таких як обробка зображень, відео, виявлення об'єктів, розпізнавання облич та багато іншого[5].

Qt є повнофункціональним фреймворком для розробки кросплатформних додатків. Він містить набір інтуїтивно зрозумілих та модульних класів бібліотеки C++, які спрощують розробку програмного забезпечення. Qt виробляє інтуїтивно зрозумілий, легко підтримуваний та повторно використовуваний код з високою продуктивністю виконання та малим розміром[6].

Для зручності написання можна скласти план:

Інтерфейс користувача (UI):

- Дві кнопки для завантаження зображень.
- Кнопка для запуску обробки.
- Вікно попереднього перегляду завантажених зображень.
- Вікно для відображення таблиці результатів.
- Кнопка для повернення в початкове меню.
- Кнопка для скачування результуючої таблиці в форматі TXT.
- Функціональні частини:

Завантаження зображень.

- Фільтрація зображень (позбавлення від шумів за допомогою морфологічних операцій).
- Визначення центрів мас точок.
- Співставлення точок на обох зображеннях.
- Відображення результатів.

Середовище розробки

В якості середовища розробки буде використано фреймворк QT на базі C++. Це зручний інструмент розробки додатків та програм на різних платформах з використанням великої кількості інтегрованих бібліотек, середовищем розробки з широким функціоналом, а також редактором розробки дизайну додату.

Qt — це кросплатформне середовище розробки програмного забезпечення, яке включає набір інструментів для створення графічних інтерфейсів користувача (GUI) та інших програм. Qt використовує мову програмування C++, але також підтримує інші мови, зокрема Python (через PyQt або PySide)[6].

Короткий опис Qt

1. **Кросплатформність**: Qt дозволяє створювати додатки для різних операційних систем, включаючи Windows, macOS, Linux, iOS, Android та інші. Код, написаний з використанням Qt, можна з легкістю переносити між цими платформами.
2. **Графічні інтерфейси**: Qt надає потужні можливості для створення сучасних графічних інтерфейсів користувача, включаючи підтримку 2D та 3D графіки.
3. **Інструменти**: Qt включає в себе власне інтегроване середовище розробки (IDE) — Qt Creator, яке забезпечує зручні засоби для написання, налагодження та тестування коду.

4. **Модульність**: Qt складається з багатьох модулів, що охоплюють різні аспекти розробки, такі як робота з мережею, базами даних, мультимедіа, потоками та інше.
5. **Ліцензування**: Qt доступний як за відкритою ліцензією (GPL, LGPL), так і за комерційною ліцензією, що робить його привабливим як для відкритих проектів, так і для комерційних розробок.

Основні плюси Qt

1. **Широкий набір функціональностей**: Qt пропонує велику кількість бібліотек і модулів, які охоплюють практично всі аспекти розробки програмного забезпечення.
2. **Потужний графічний інтерфейс**: Qt має розвинуті засоби для створення користувацьких інтерфейсів з високим рівнем деталізації та інтерактивності.
3. **Активна спільнота і підтримка**: Завдяки великій та активній спільноті розробників, Qt має багато ресурсів для навчання, прикладів, бібліотек та інструментів.
4. **Модульність та розширюваність**: Модульна архітектура Qt дозволяє легко додавати нові можливості та інтегрувати сторонні бібліотеки.
5. **Підтримка сучасних технологій**: Qt постійно оновлюється і підтримує новітні стандарти та технології, включаючи інтеграцію з іншими мовами програмування та інструментами.
6. **Висока продуктивність**: Завдяки використанню C++ Qt забезпечує високу швидкість виконання додатків і ефективне використання ресурсів.

2.2.1 Алгоритм обробки зображень

1. **Завантаження зображень:**
   - Використовується QFileDialog для вибору файлів зображень у форматі BMP.

- Завантаження зображень за допомогою OpenCV функції *cv::imread()*.
- Якщо зображення не завантажене, відображається повідомлення про помилку.

2. **Відображення зображень:**
- Зображення конвертується у формат RGB за допомогою *cv::cvtColor()*.
- Відображення зображення у QLabel за допомогою QPixmap.

3. **Обробка зображень:**
- Зображення обробляються для видалення шуму за допомогою Гаусового розмиття та морфологічних операцій (ерозія та дилатація).
- Знаходяться контури на зображенні та обчислюються центри мас для кожного контуру.
- Співставляються точки з двох зображень на основі мінімальної відстані.

4. **Відображення результатів:**
- Результати обробки відображаються у QTableWidget у вигляді таблиці з координатами точок та різницями координат.

5. **Збереження результатів:**
- Результати зберігаються у текстовий файл у форматі TXT за допомогою QFile та QTextStream.

2.2.2 Структура проекту

Структура проекту складається з п'яти файлів:

Diplom/

├──Diplom.pro

├── main.cpp

├── mainwindow.h

├── mainwindow.cpp

└── mainwindow.ui

Короткий опис кожного файлу:

YourProjectName.pro:

- Це файл конфігурації проекту Qt, який визначає параметри збірки, залежності та шляхи до бібліотек.

main.cpp:

- Головний файл програми, який ініціалізує додаток Qt і відкриває головне вікно.

mainwindow.h:

- Заголовковий файл для класу *MainWindow*, який оголошує змінні та методи, що використовуються у головному вікні програми.

mainwindow.cpp:

- Реалізаційний файл для класу *MainWindow*, який містить код для обробки зображень, взаємодії з інтерфейсом користувача та збереження результатів.

mainwindow.ui:

- XML файл, створений за допомогою Qt Designer, який описує графічний інтерфейс користувача (GUI).

Ці файли є класичним набором розробки додатку в QT. Всі вони будуть знаходитись в папці проекту. Далі буде наведений детальний опис кожного файлу, а також код кожної частини розробки програми. Також в проект додані коментарі для зручного читання та пояснення коду.

Файл YourProjectName.pro

Файл конфігурації проекту, який описує, які бібліотеки та модулі будуть використовуватись:

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = YourProjectName
TEMPLATE = app

SOURCES += main.cpp \
           mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

# Додайте шляхи до OpenCV
INCLUDEPATH += C:/cv/opencv/build/include
LIBS += -LC:/cv/opencv/build/x64/vc16/lib \
        -lopencv_world4100d
LIBS += -LC:/cv/opencv/build/x64/vc16/bin \
        -lopencv_world4100d
```

Рисунок 2.2 Код YourProjectName.pro

Перш дві строчки відносяться до перевірки версії qt та є обов'язковими для нормального функціонування проекту.

Строчки з 4 до 12 є описом конфігурації проекту. Вона описана в структурі проекту.

Строчки з 15 по 19 підключають бібліотеку opencv.

Файл main.cpp

Головний файл програми, який ініціалізує додаток та відображає головне вікно:

```cpp
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```

Рисунок 2.3 Код main.cpp

Він включає заголовочний файл QApplication і файл заголовка "mainwindow.h". Функція main() є головною функцією програми, яка створює об'єкт класу QApplication з передачею аргументів командного рядка argc і argv. Потім створюється об'єкт класу MainWindow, який є головним вікном програми, і він відображається на екрані за допомогою методу show(). На останок, функція exec() об'єкта QApplication виконує головний цикл обробки подій та повертає управління після закриття вікна.

Файл mainwindow.h

Заголовковий файл для головного вікна, який оголошує всі необхідні змінні та функції:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QLabel>
#include <QMainWindow>
#include <QTableWidget>
#include <opencv2/opencv.hpp>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
public slots: // Оголошення слотів
    void on_loadButton1_clicked();
    void on_loadButton2_clicked();
    void on_processButton_clicked();
    void on_backButton_clicked(); // Оголошення методу для кнопки "Назад"
    void on_saveButton_clicked();
    // Оголошення інших слотів
```

Рисунок 2.4 Частина коду mainwindow.h

Цей код є заголовковим файлом для класу MainWindow, який є підкласом QMainWindow від Qt. Він містить оголошення конструктора, деструктора, приватних слотів (private slots) для обробки подій кнопок, приватних методів для відображення зображень, обробки зображень та збереження результатів у файл, а також приватне поле для збереження відповідностей між точками на зображеннях. Також включає необхідні бібліотеки для роботи з OpenCV і Qt.

Файл mainwindow.cpp

Реалізація функцій головного вікна, включаючи завантаження, обробку та збереження зображень(код в силу великого об'єму винесений в додаток).

Цей код є частиною класу MainWindow, який є частиною графічного інтерфейсу програми на Qt. Клас містить слоти (on_loadButton1_clicked та

on_loadButton2_clicked), які викликаються при натисканні відповідних кнопок на інтерфейсі.

При натисканні кнопки "Load Image 1" (loadButton1), відбувається відкриття вікна вибору файлу за допомогою QFileDialog. Після вибору файлу, відбувається завантаження зображення за допомогою OpenCV (cv::imread) у відтінках сірого. Якщо завантаження пройшло успішно, зображення відображається на інтерфейсі за допомогою функції displayImage. У разі невдачі завантаження, виводиться повідомлення про помилку за допомогою QMessageBox та виводиться відладочна інформація у консоль.

Аналогічно, при натисканні кнопки "Load Image 2" (loadButton2), відбувається аналогічний процес завантаження та відображення другого зображення.

Цей код також містить підключення необхідних бібліотек (mainwindow.h, ui_mainwindow.h, QFileDialog, QMessageBox, QTableWidget, QFile, QTextStream, opencv2/opencv.hpp, iostream) та конструктор/деструктор класу.

Пояснення функцій:

1. MainWindow::MainWindow та MainWindow::~MainWindow:

   Конструктор та деструктор класу MainWindow. Конструктор ініціалізує інтерфейс користувача, деструктор видаляє об'єкт інтерфейсу користувача.

2. void MainWindow::on_loadButton1_clicked():

   Викликається при натисканні кнопки Load Image 1.

   Відкриває діалогове вікно вибору файлу та завантажує вибране зображення у змінну image1.

   Якщо зображення успішно завантажене, воно відображається у QLabel labelImage1.

3. void MainWindow::on_loadButton2_clicked():

   Аналогічна функція для завантаження другого зображення у змінну image2.

4. void MainWindow::on_processButton_clicked():

Викликається при натисканні кнопки Process.

Якщо обидва зображення завантажені, викликає функцію processImages().

5. void MainWindow::on_saveButton_clicked():

Викликається при натисканні кнопки Save Results.

Відкриває діалогове вікно збереження файлу та зберігає результати у вибраний файл.

6. void MainWindow::displayImage(const cv::Mat &image, QLabel *label):

Перетворює зображення з формату OpenCV на формат RGB.

Відображає зображення у вказаному QLabel.

7. void MainWindow::processImages():

Обробляє обидва зображення, видаляє шум, знаходить центри мас та співставляє точки.

Відображає результати у таблиці.

8. cv::Mat MainWindow::denoise_image(const cv::Mat& image):

Застосовує Гаусове розмиття та морфологічні операції для видалення шуму з зображення.

9. std::vector<cv::Point2f> MainWindow::find_centroids(const cv::Mat& image):

Знаходить контури на зображенні та обчислює центри мас для кожного контуру.

10. std::vector<std::pair<cv::Point2f, cv::Point2f>> MainWindow:: match_points (const std::vector < cv::Point2f>& points1, const std::vector < cv::Point2f> & points2):

Співставляє точки з двох зображень на основі мінімальної відстані.

11. void MainWindow: :display_results (const std: :vector< std::pair<cv ::Point2f, cv ::Point2f>>& matches, QTableWidget* table):

Відображає результати обробки у таблиці з шістьма стовпчиками.

12. void MainWindow::saveResultsToFile(const QString& fileName):

Зберігає результати обробки у текстовий файл.

Файл mainwindow.ui

XML-код для файлу mainwindow.ui, який описує інтерфейс користувача(код в силу великого об'єму винесений в додаток).

Цей код є описом графічного інтерфейсу користувача (GUI) за допомогою мови розмітки XML для використання в програмі, що використовує бібліотеку Qt. Він описує вікно програми (QMainWindow) з різними віджетами, такими як кнопки (QPushButton) і мітки (QLabel), а також їхню геометрію (розміщення та розмір), текст, який вони відображають, та інші властивості.

Цей код також включає в себе розміщення та розмір головного вікна програми, його заголовок та версію інтерфейсу.

XML-код використовується для створення графічного інтерфейсу за допомогою Qt-бібліотеки, який буде містити кнопки для завантаження зображень, обробки зображень, повернення до меню та збереження результатів.

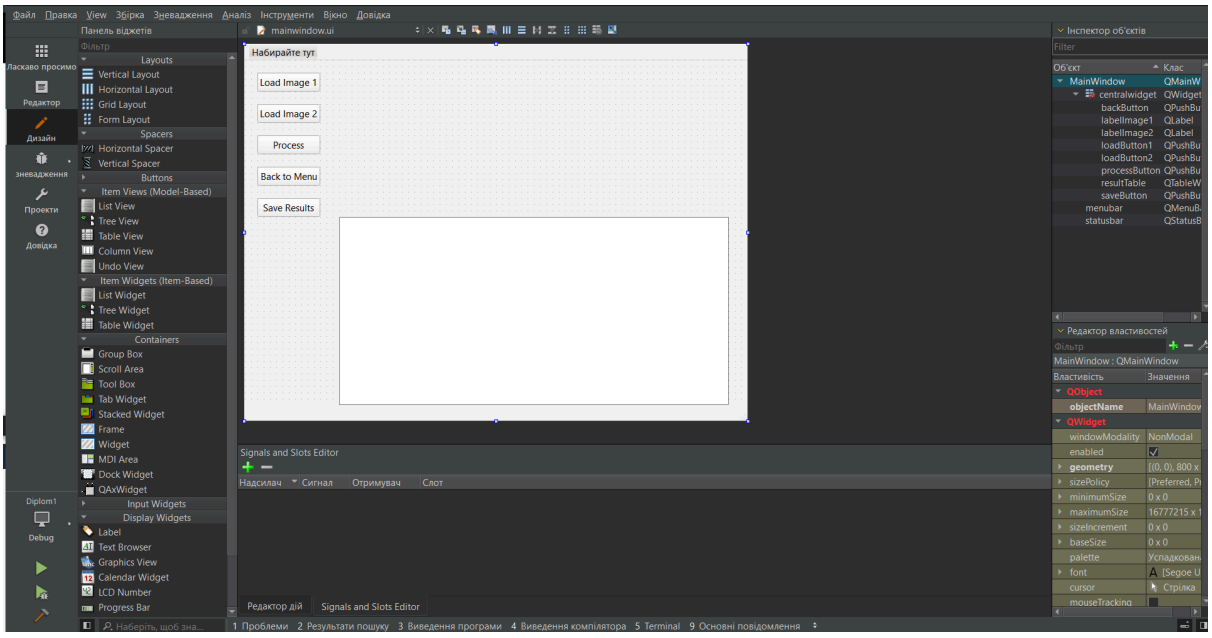


Рисунок 2.5 Інтерфейс mainwindow.ui

2.2.3 Інструкція по використанні програми

Як користуватись програмою:

- Запустіть програму.

- Завантажте зображення:

Натисніть кнопку "Load Image 1" та виберіть перше зображення у форматі BMP (Рисунок 2.6).

Натисніть кнопку "Load Image 2" та виберіть друге зображення у форматі BMP(Рисунок 2.6).

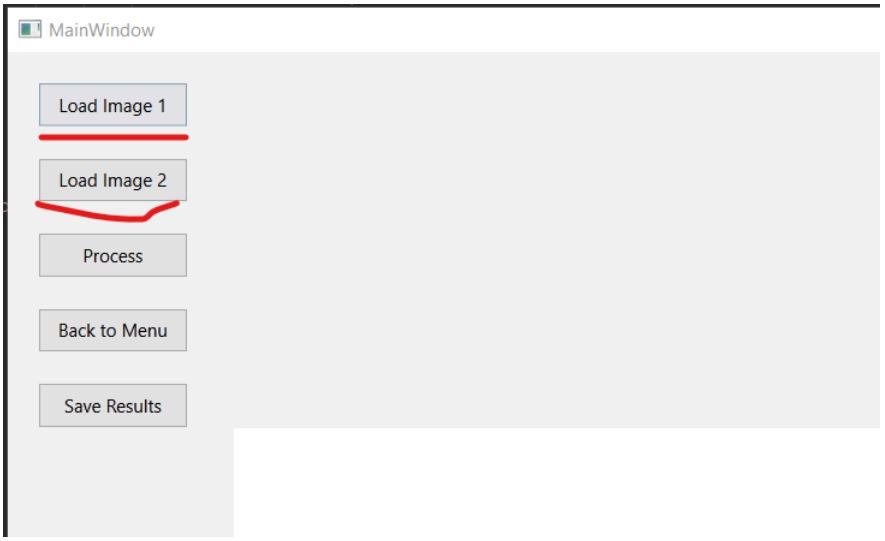


Рисунок 2.6 Інтерфейс програми.

- Обробіть зображення:

Натисніть кнопку "Process", щоб обробити зображення, знайти центри мас точок та співставити їх(Рисунок 2.7).

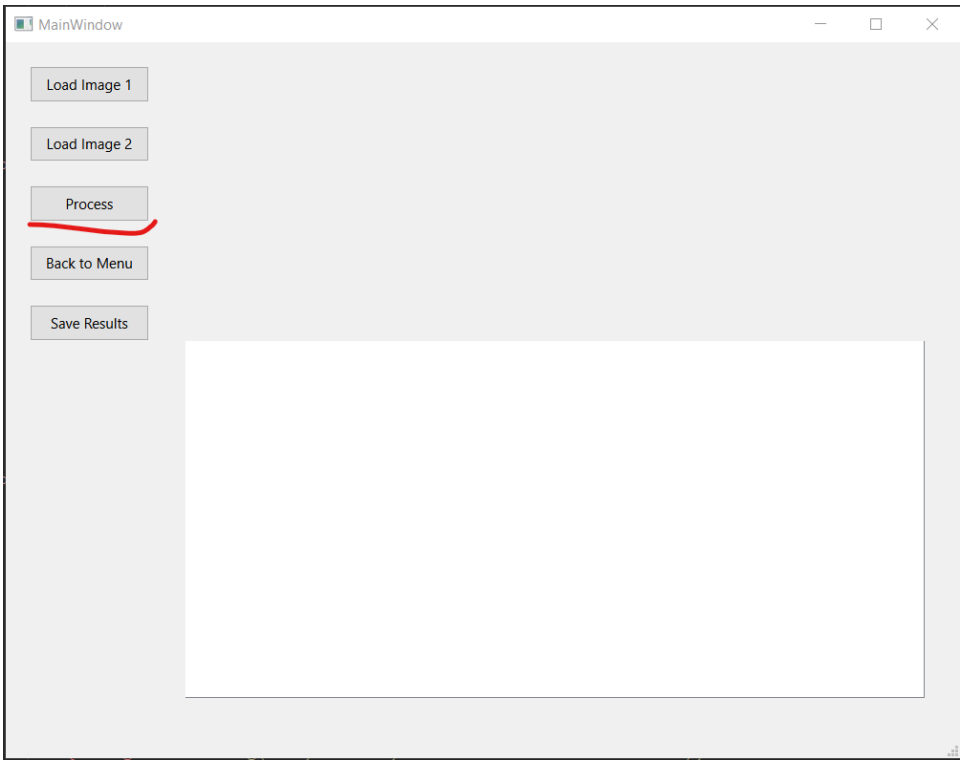


Рисунок 2.7 Кнопка запуску обчислень

- Перегляньте результати:

Результати обробки будуть відображені у таблиці.

- Збережіть результати:

Натисніть кнопку "Save Results" для збереження результатів у файл TXT.

Цей код забезпечує завантаження, обробку та збереження результатів аналізу зображень, використовуючи бібліотеки Qt для інтерфейсу користувача та OpenCV для обробки зображень.

РОЗДІЛ 3. ТЕСТУВАННЯ РОБОТОЗДАТНОСТІ ПРОГРАМИ

Протестуємо декілька випадків.

3.1 Перевірка працездатності програми

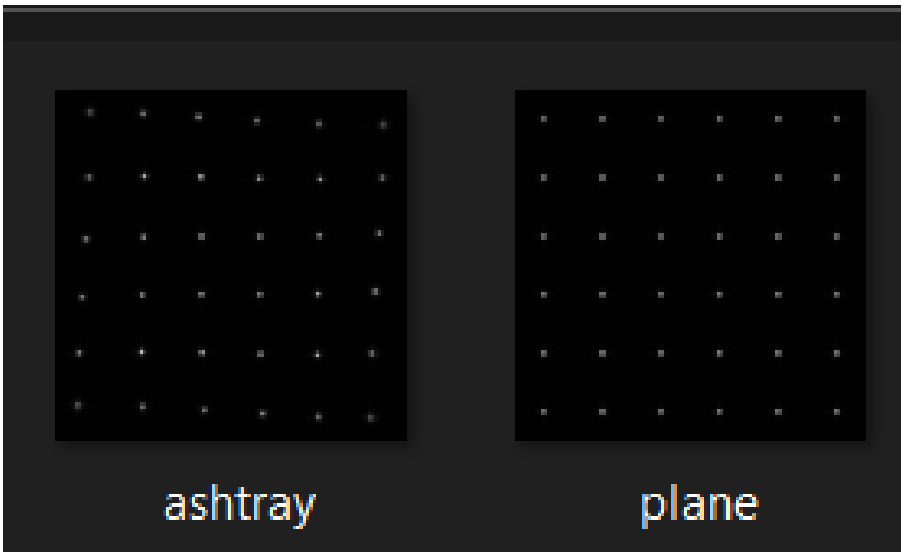
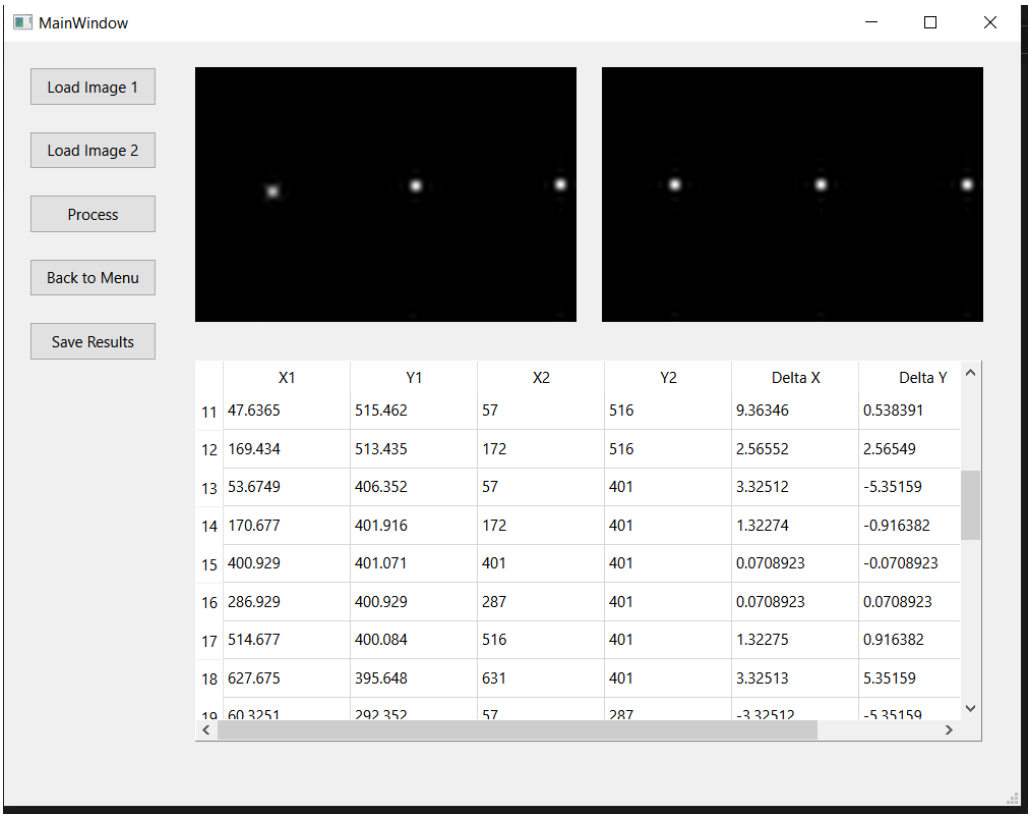


Рисунок 3.1 Набір файлів

Прикладом буде два зображення з точками 6 на 6. Точки по сторонам затемнені.



	X1	Y1	X2	Y2	Delta X	Delta Y
11	47.6365	515.462	57	516	9.36346	0.538391
12	169.434	513.435	172	516	2.56552	2.56549
13	53.6749	406.352	57	401	3.32512	-5.35159
14	170.677	401.916	172	401	1.32274	-0.916382
15	400.929	401.071	401	401	0.0708923	-0.0708923
16	286.929	400.929	287	401	0.0708923	0.0708923
17	514.677	400.084	516	401	1.32275	0.916382
18	627.675	395.648	631	401	3.32513	5.35159
19	60.3251	292.352	57	287	-3.32512	-5.35159

Рисунок 3.2 Результат роботи програми

Як видно на Рисунку 3.2 програма успішно запустилась та виконала поставлену задачу.

Таблиця 3.1 Результуюча таблиця

X1	Y1	X2	Y2	Delta X	Delta Y
515.462	640.363	516	631	0.538391	-9.36346
619.032	642.968	631	631	11.968	-11.968
406.352	634.325	401	631	-5.35159	-3.32513
292.352	627.675	287	631	-5.35159	3.32513
171.462	621.637	172	631	0.538391	9.36346
45.032	619.032	57	631	11.968	11.968
513.435	518.565	516	516	2.56549	-2.56549
401.916	517.323	401	516	-0.916382	-1.32275
621.637	516.538	631	516	9.36346	-0.538391
287.916	514.677	287	516	-0.916382	1.32275
47.6365	515.462	57	516	9.36346	0.538391
169.434	513.435	172	516	2.56552	2.56549
53.6749	406.352	57	401	3.32512	-5.35159
170.677	401.916	172	401	1.32274	-0.916382
400.929	401.071	401	401	0.0708923	-0.0708923
286.929	400.929	287	401	0.0708923	0.0708923
514.677	400.084	516	401	1.32275	0.916382
627.675	395.648	631	401	3.32513	5.35159
60.3251	292.352	57	287	-3.32512	-5.35159
173.323	287.916	172	287	-1.32274	-0.916382
401.071	287.071	401	287	-0.0708923	-0.0708923
287.071	286.929	287	287	-0.0708923	0.0708923
517.323	286.084	516	287	-1.32275	0.916382
634.325	281.648	631	287	-3.32513	5.35159
518.565	174.566	516	172	-2.56549	-2.56552
400.084	173.323	401	172	0.916382	-1.32274
640.363	172.538	631	172	-9.36346	-0.538391
286.084	170.677	287	172	0.916382	1.32274
66.3635	171.462	57	172	-9.36346	0.538391
174.566	169.434	172	172	-2.56552	2.56552
516.538	66.3635	516	57	-0.538391	-9.36346
642.968	68.968	631	57	-11.968	-11.968
395.648	60.3251	401	57	5.35159	-3.32512
281.648	53.6749	287	57	5.35159	3.32512
172.538	47.6365	172	57	-0.538391	9.36346
68.968	45.032	57	57	-11.968	11.968

Звіряємо результат роботи програми(таблиця 3.1) з програмою перевірки. Дані співпадають.

3.2 Перевірка працездатності програми

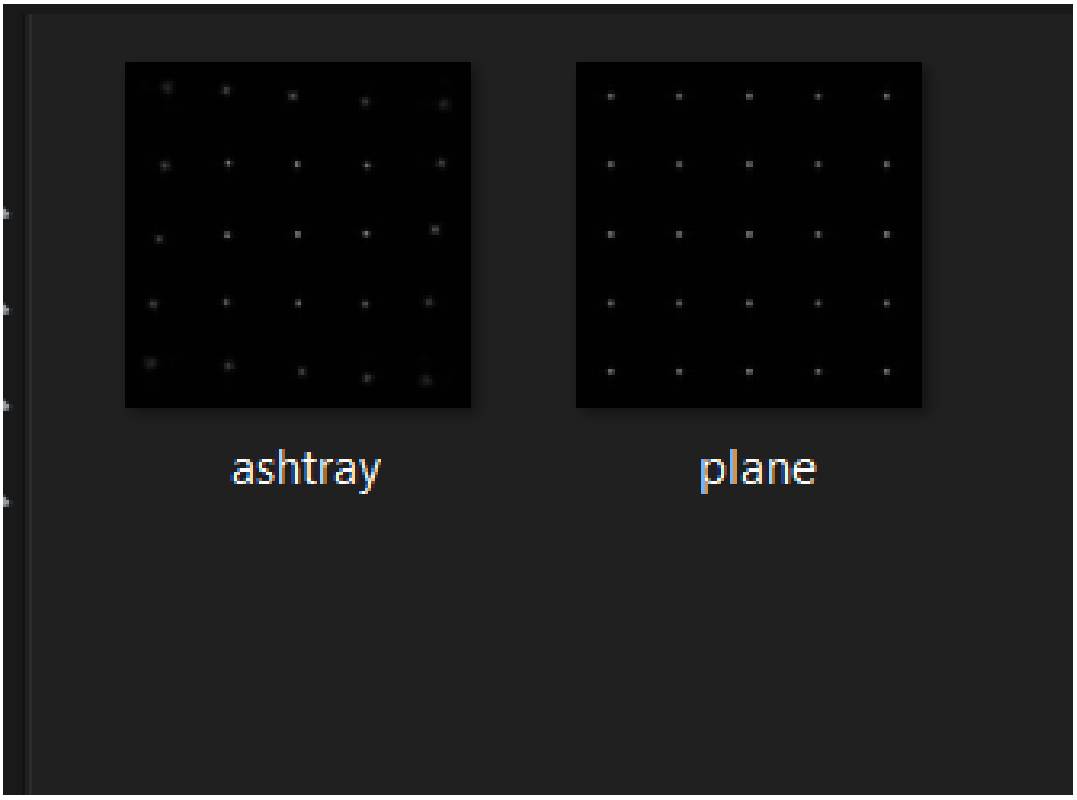


Рисунок 3.3 Набір файлів

Другим прикладом буде два зображення з точками 5 на 5. Точки по сторонам набагато більш затемнені та розмиті. Це збільшує небезпеку втрати точок.

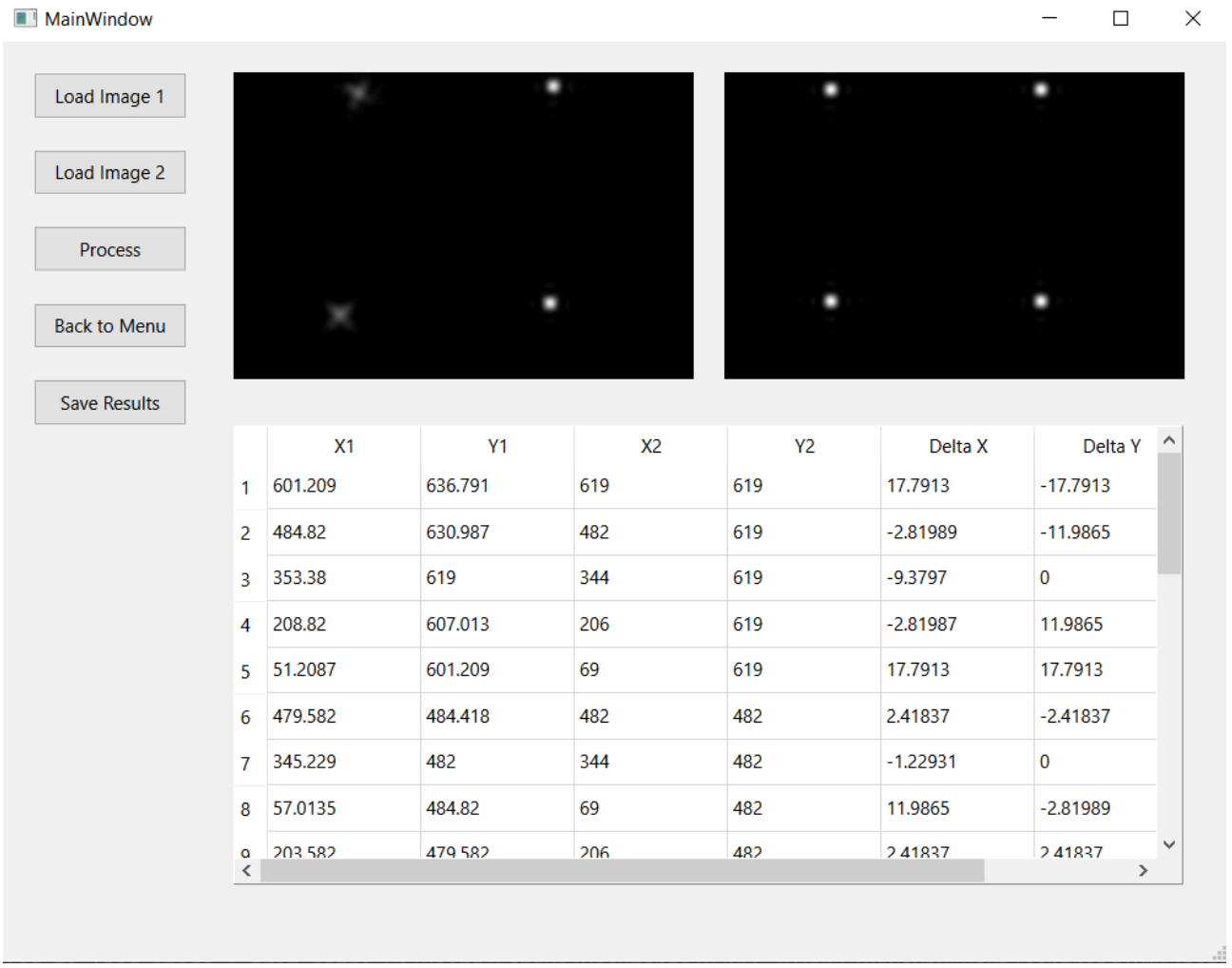


Рисунок 3.4 Результат роботи програми

Як видно на Рисунку 3.2 програма успішно запустилась та виконала поставлену задачу.

Таблиця 3.2 Результуюча таблиця

X1	Y1	X2	Y2	Delta X	Delta Y
601.209	636.791	619	619	17.7913	-17.7913
484.82	630.987	482	619	-2.81989	-11.9865
353.38	619	344	619	-9.3797	0
208.82	607.013	206	619	-2.81987	11.9865
51.2087	601.209	69	619	17.7913	17.7913
479.582	484.418	482	482	2.41837	-2.41837
345.229	482	344	482	-1.22931	0
57.0135	484.82	69	482	11.9865	-2.81989
203.582	479.582	206	482	2.41837	2.41837
607.013	479.18	619	482	11.9865	2.81989
69	353.38	69	344	0	-9.3797
206	345.229	206	344	0	-1.22931
344	344	344	0	0	
482	342.771	482	344	0	1.22931
619	334.62	619	344	0	9.3797
484.418	208.418	482	206	-2.41837	-2.41837
342.771	206	344	206	1.22931	0
80.9865	208.82	69	206	-11.9865	-2.81987
208.418	203.582	206	206	-2.41837	2.41837
630.987	203.18	619	206	-11.9865	2.81987
636.791	86.7913	619	69	-17.7913	-17.7913
479.18	80.9865	482	69	2.81989	-11.9865

334.62	69	344	69	9.3797	0
203.18	57.0135		206	69	2.81987 11.9865
86.7913	51.2087		69	69	-17.791317.7913

Як видно по таблиці, програма успішно знайшла всі 25 точок. Звіряємо результат роботи програми(таблиця 3.2) з програмою перевірки. Дані співпадають.

ВИСНОВОК

У процесі розробки даної програми було досягнуто кілька ключових результатів:

Застосування технологій: Було використано сучасні інструменти та бібліотеки, такі як Qt для створення графічного інтерфейсу користувача та OpenCV для обробки зображень. Це дозволило забезпечити високу продуктивність і гнучкість програми.

Реалізація функціональності: Програма успішно реалізує функції завантаження, обробки та відображення зображень. Користувач може завантажити два зображення, обробити їх, знайти центроїди об'єктів на зображеннях та зіставити їх. Результати обробки відображаються у вигляді таблиці та можуть бути збережені у текстовий файл.

Модульність і масштабованість: Структура програми розроблена таким чином, що легко можна додавати нові функції. Всі основні компоненти, такі як обробка зображень, пошук центроїдів, зіставлення точок та збереження результатів, реалізовані у вигляді окремих функцій, що спрощує їх модифікацію та розширення.

Інтерфейс користувача: Завдяки використанню Qt вдалося створити інтуїтивно зрозумілий інтерфейс, який забезпечує зручність у використанні. Користувач має можливість завантажувати зображення, натискати кнопки для запуску обробки та збереження результатів.

Практична цінність: Розроблена програма може бути використана для обробки даних з датчика Шека–Гартмана, їх цифровізації та збережені, що є корисним в багатьох галузях, таких як адаптивна оптика, астрономія, офтальмологія та інших галузях, де потрібна висока точність вимірювання оптичних хвильових фронтів.

ДЖЕРЕЛА

1. Robert C.Cannon Global wave-front reconstruction using Shack-Hartmann sesors // J.Opt.Soc.Am.A Vol. 12 No.9 Sept 1995
2. Матриця мікролінз. URL: https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/full/10.1002/jemt.23818 (дата звернення: 19.05.2024)
3. Алгоритми обробки зображень. URL: https://neptune.ai/blog/image-processing-python (дата звернення: 19.05.2024)
4. Гаусове згладження. URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm (дата звернення: 18.05.2024)
5. Adrian Kaehler, Gary Bradski Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library // O'Reilly 2016
6. Документація Qt. URL: https://doc.qt.io (дата звернення: 21.05.2024)

ДОДАТОК

Лістинг програми обробки гартманограм

Diplom1_1.pro

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = YourProjectName
TEMPLATE = app
SOURCES += main.cpp \
           mainwindow.cpp
HEADERS  += mainwindow.h
FORMS    += mainwindow.ui
INCLUDEPATH += C:/cv/opencv/build/include
LIBS += -LC:/cv/opencv/build/x64/vc16/lib \
        -lopencv_world4100d
LIBS += -LC:/cv/opencv/build/x64/vc16/bin \
        -lopencv_world4100d
```

main.cpp

```
#include <QApplication>
#include "mainwindow.h"
int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QFileDialog>
#include <QMessageBox>
#include <QTableWidget>
#include <QFile>
#include <QTextStream>
#include <opencv2/opencv.hpp>
#include <iostream>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_loadButton1_clicked()
{
    QString filePath = QFileDialog::getOpenFileName(this, "Open Image 1", "", "BMP Files (*.bmp)");
    if (!filePath.isEmpty()) {
        image1 = cv::imread(filePath.toStdString(), cv::IMREAD_GRAYSCALE);
        if (image1.empty()) {
            QMessageBox::warning(this, "Error", "Failed to load image 1.");
                std::cerr << "Error: Failed to load image 1 from path: " << filePath.toStdString() << std::endl; //
Відладочна інформація
        } else {
            displayImage(image1, ui->labelImage1);
        }
    } else {
        std::cerr << "Error: No file selected for image 1." << std::endl; // Відладочна інформація
```

```
    }
}

void MainWindow::on_loadButton2_clicked()
{
    QString filePath = QFileDialog::getOpenFileName(this, "Open Image 2", "", "BMP Files (*.bmp)");
    if (!filePath.isEmpty()) {
        image2 = cv::imread(filePath.toStdString(), cv::IMREAD_GRAYSCALE);
        if (image2.empty()) {
            QMessageBox::warning(this, "Error", "Failed to load image 2.");
                std::cerr << "Error: Failed to load image 2 from path: " << filePath.toStdString() << std::endl; // Відладочна інформація
        } else {
            displayImage(image2, ui->labelImage2);
        }
    } else {
        std::cerr << "Error: No file selected for image 2." << std::endl; // Відладочна інформація
    }
}

void MainWindow::on_processButton_clicked()
{
    if (!image1.empty() && !image2.empty()) {
        processImages();
    } else {
        QMessageBox::warning(this, "Error", "Please load both images before processing.");
        std::cerr << "Error: One or both images are not loaded." << std::endl;
    }
}

void MainWindow::on_backButton_clicked()
{
    // Повернення до меню
}

void MainWindow::on_saveButton_clicked()
{
```

```
    QString fileName = QFileDialog::getSaveFileName(this, "Save Results", "", "Text Files (*.txt)");
    if (!fileName.isEmpty()) {
        saveResultsToFile(fileName);
    }
}

void MainWindow::displayImage(const cv::Mat &image, QLabel *label) {
    cv::Mat rgb;
    cv::cvtColor(image, rgb, cv::COLOR_GRAY2RGB);
    QImage qimg(rgb.data, rgb.cols, rgb.rows, rgb.step, QImage::Format_RGB888);
    label->setPixmap(QPixmap::fromImage(qimg));
}

std::vector<cv::Point2f> MainWindow::find_centroids(const cv::Mat& image) {
    cv::Mat gray, binary;
    std::vector<cv::Point2f> centroids;

    // Перетворення в градації сірого, якщо зображення не в градаціях сірого
    if (image.channels() > 1) {
        cv::cvtColor(image, gray, cv::COLOR_BGR2GRAY);
    } else {
        gray = image;
    }

    // Бінаризація зображення
    cv::threshold(gray, binary, 14, 255, cv::THRESH_BINARY);

    // Знаходження контурів
    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(binary, contours, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);

    // Обчислення центрів мас
    for (const auto& contour : contours) {
        cv::Moments m = cv::moments(contour);
        if (m.m00 != 0) {
            centroids.push_back(cv::Point2f(m.m10 / m.m00, m.m01 / m.m00));
        }
```

```
    }


    return centroids;
}


std::vector<std::pair<cv::Point2f, cv::Point2f>> MainWindow::match_points(const std::vector<cv::Point2f>& points1, const std::vector<cv::Point2f>& points2) {
    std::vector<std::pair<cv::Point2f, cv::Point2f>> matches;
    for (const auto& p1 : points1) {
        float minDist = std::numeric_limits<float>::max();
        cv::Point2f bestMatch;
        for (const auto& p2 : points2) {
            float dist = cv::norm(p1 - p2);
            if (dist < minDist) {
                minDist = dist;
                bestMatch = p2;
            }
        }
        matches.push_back({p1, bestMatch});
    }
    return matches;
}


void MainWindow::display_results(const std::vector<std::pair<cv::Point2f, cv::Point2f>>& matches, QTableWidget* table) {
    table->setRowCount(matches.size());
    table->setColumnCount(6);
    table->setHorizontalHeaderLabels({"X1", "Y1", "X2", "Y2", "Delta X", "Delta Y"});


    for (int i = 0; i < matches.size(); ++i) {
        table->setItem(i, 0, new QTableWidgetItem(QString::number(matches[i].first.x)));
        table->setItem(i, 1, new QTableWidgetItem(QString::number(matches[i].first.y)));
        table->setItem(i, 2, new QTableWidgetItem(QString::number(matches[i].second.x)));
        table->setItem(i, 3, new QTableWidgetItem(QString::number(matches[i].second.y)));
        table->setItem(i, 4, new QTableWidgetItem(QString::number(matches[i].second.x - matches[i].first.x)));
        table->setItem(i, 5, new QTableWidgetItem(QString::number(matches[i].second.y - matches[i].first.y)));
    }
}
```

```
void MainWindow::processImages() {
    cv::Mat denoised1 = denoise_image(image1);
    cv::Mat denoised2 = denoise_image(image2);

    std::vector<cv::Point2f> centroids1 = find_centroids(denoised1);
    std::vector<cv::Point2f> centroids2 = find_centroids(denoised2);

    matches = match_points(centroids1, centroids2);

    display_results(matches, ui->resultTable);
}

cv::Mat MainWindow::denoise_image(const cv::Mat& image) {
    cv::Mat denoised;
    cv::GaussianBlur(image, denoised, cv::Size(5, 5), 2);
    cv::Mat morph;
    cv::erode(denoised, morph, cv::Mat(), cv::Point(-1, -1), 2);
    cv::dilate(morph, morph, cv::Mat(), cv::Point(-1, -1), 2);
    return morph;
}

void MainWindow::saveResultsToFile(const QString& fileName) {
    QFile file(fileName);
    if (file.open(QIODevice::WriteOnly)) {
        QTextStream stream(&file);
        stream << "X1\tY1\tX2\tY2\tDelta X\tDelta Y\n";
        for (const auto& match : matches) {
            stream << match.first.x << "\t" << match.first.y << "\t"
                   << match.second.x << "\t" << match.second.y << "\t"
                   << match.second.x - match.first.x << "\t"
                   << match.second.y - match.first.y << "\n";
        }
        file.close();
    } else {
        QMessageBox::warning(this, "Error", "Failed to save the results.");
    }
```

```
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QLabel>
#include <QMainWindow>
#include <QTableWidget>
#include <opencv2/opencv.hpp>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
public slots: // Оголошення слотів
    void on_loadButton1_clicked();
    void on_loadButton2_clicked();
    void on_processButton_clicked();
    void on_backButton_clicked(); // Оголошення методу для кнопки "Назад"
    void on_saveButton_clicked();
    // Оголошення інших слотів

private:
    Ui::MainWindow *ui;
    cv::Mat image1;
    cv::Mat image2;

    void displayImage(const cv::Mat &image, QLabel *label);
```

```
    void processImages();

    void saveResultsToFile(const QString& fileName);

    cv::Mat denoise_image(const cv::Mat& image);

    std::vector<std::pair<cv::Point2f, cv::Point2f>> matches;

    std::vector<cv::Point2f> find_centroids(const cv::Mat& image);

        std::vector<std::pair<cv::Point2f, cv::Point2f>> match_points(const std::vector<cv::Point2f>& points1, const std::vector<cv::Point2f>& points2); // Оголошення методу match_points

        void display_results(const std::vector<std::pair<cv::Point2f, cv::Point2f>>& matches, QTableWidget* table);

};

#endif // MAINWINDOW_H
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="loadButton1">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>20</y>
      <width>100</width>
      <height>30</height>
     </rect>
    </property>
    <property name="text">
     <string>Load Image 1</string>
    </property>
   </widget>
   <widget class="QPushButton" name="loadButton2">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>70</y>
      <width>100</width>
```

```
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>Load Image 2</string>
 </property>
</widget>
<widget class="QPushButton" name="processButton">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>120</y>
   <width>100</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>Process</string>
 </property>
</widget>
<widget class="QPushButton" name="backButton">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>170</y>
   <width>100</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>Back to Menu</string>
 </property>
</widget>
<widget class="QPushButton" name="saveButton">
 <property name="geometry">
  <rect>
   <x>20</x>
```

```
   <y>220</y>
   <width>100</width>
   <height>30</height>
  </rect>
 </property>
 <property name="text">
  <string>Save Results</string>
 </property>
</widget>
<widget class="QLabel" name="labelImage1">
 <property name="geometry">
  <rect>
   <x>150</x>
   <y>20</y>
   <width>300</width>
   <height>200</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="labelImage2">
 <property name="geometry">
  <rect>
   <x>470</x>
   <y>20</y>
   <width>300</width>
   <height>200</height>
  </rect>
 </property>
</widget>
<widget class="QTableWidget" name="resultTable">
 <property name="geometry">
  <rect>
   <x>150</x>
   <y>250</y>
   <width>620</width>
   <height>300</height>
  </rect>
```

```
   </property>
  </widget>
 </widget>
 <widget class="QMenuBar" name="menubar">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>25</height>
   </rect>
  </property>
 </widget>
 <widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```