МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАТИКИ ТА КОМП'ЮТЕРНОЇ ТЕХНІКИ

Рівень вищої освіти	Перший (бакалаврський)
Спеціальність	Інформаційні системи та технології
Освітня програма	Інформаційні системи та технології
Група	6.04.126.010.19.1

ДИПЛОМНИЙ ПРОЄКТ

на тему: «Розроблення модуля інформаційної системи для пошуку нерегулярних частин зображення»

Виконав: студент Тимур ЗАДІКЯН

Керівник: к.т.н, доц. Олексій ГОРОХОВАТСЬКИЙ

Рецензент: к.т.н, доц. кафедри Інформатики
Харківського національного університету радіоелектроніки
доц. Валентин ЛЮБЧЕНКО

Харків – 2023 рік

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Факультет Інформаційних технологій
Кафедра Інформатики та комп'ютерної техніки
Освітній ступінь Бакалавр
Спеціальність 126 "Інформаційні системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інформатики та комп'ютерної техніки
Проф. Сергій УДОВЕНКО
"01" лютого 2023 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ
Задікян Тимур Едуардовича

1. Тема проєкту: «Розроблення модуля інформаційної системи для пошуку нерегулярних частин зображення»

керівник проєкту: Доцент кафедри ІКТ, к.т.н, доц. Гороховатський Олексій Володимирович

затверджені наказом ректора від "01 лютого" 2023 р. № 107-С

2. Строк подання студентом проєкту: 01 червня 2023 року

3. Вихідні дані до проєкту: опис методів пошуку нерегулярних частин на зображеннях, аналіз модулів обробки нерегулярних частин на зображеннях, розроблення модуля інформаційної системи для пошуку нерегулярних частин на зображеннях, розроблення інтерфейсу модуля для застосунку, аналіз результатів моделювання.

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

Розділ 1. Методи для пошуку нерегулярних частин на зображеннях.

Розділ 2. Аналіз модулів обробки нерегулярних частин на зображеннях.

Розділ 3. Розроблення модуля інформаційної системи для пошуку нерегулярних частин на зображеннях

5. Перелік графічного матеріалу: актуальність проблеми (1 слайд), мета та завдання проєктування (1 слайд), опис методів пошуку нерегулярних частин на зображеннях (2-3 слайди), дослідження точності та ефективності реалізованої моделі (2-3 слайди), розроблення застосунку (3-4 слайди), результати порівняння із аналогами (1 слайд), висновки (1 слайд).

6. Консультація розділів дипломного проєкту. Консультант відсутній.

7. Дата видачі завдання: "01" лютого 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Розроблення плану дипломного проєкту, ознайомлення з літературними джерелами за темою.	01.02.2023 – 28.02.2023	
2.	Аналіз предметної області.	01.02.2023 – 28.02.2023	
3.	Дослідження методів обробки зображень.	01.03.2023 – 01.04.2023	
4.	Розроблення застосунку для обробки зображень.	01.03.2023 – 20.05.2023	
5.	Перевірка чернетки дипломного проєкту та внесення змін до неї керівником.	01.04.2023 – 10.04.2023	
6.	Оформлення дипломного проєкту, перевірка якості дипломного проєкту на плагіат.	01.05.2023 - 31.05.2023	
7.	Подання Голові Екзаменаційної комісії щодо захисту дипломного проєкту.	01.06.2023	

Студент Тимур Задікян

Керівник проєкту Олексій Гороховатський

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту: 43 сторінки, 38 рисунків, 25 джерел.

Об'єктом розробки є модуль інформаційної системи для пошуку нерегулярних частин зображення, який включає застосунок та пов'язані з ним функціональність та компоненти.

Метою дипломного проекту є розроблення застосунку для пошуку нерегулярних частин зображення.

Початковий етап дослідження включає збір та огляд наукових публікацій, академічних досліджень і комерційних продуктів, пов'язаних з обробкою зображень та аналізом нерегулярних частин. Далі проводиться аналіз і класифікація різних методів обробки зображень, що використовуються для пошуку нерегулярних частин. Цей аналіз включає вивчення алгоритмів, методів комп'ютерного зору, машинного навчання та інших відповідних технік, що можуть бути застосовані у контексті пошуку нерегулярних частин на зображеннях.

Після цього вибирається найбільш підходящий підхід для реалізації модуля пошуку нерегулярних частин. Цей вибір здійснюється на основі критеріїв ефективності, точності та швидкодії, а також з урахуванням вимог технічних засобів розробки. Після вибору методу розробляється програмний модуль, що реалізує обраний метод.

ІНФОРМАЦІЙНА СИСТЕМА, МЕТОД РОЗРОБКИ, КОМ’ЮТЕРНИЙ ЗІР, НЕРЕГУЛЯРНІ ЧАСТИНИ.

ABSTRACT

Explanatory note to the diploma project: 43 pages, 38 figures, 25 sources.

The object of development is the module of an information system for searching for irregular parts on the image", which includes an application and related functionality and components.

The goal of the project is to develop an application for searching of the irregular parts of an image.

The initial stage of the research includes collecting and review of scientific publications, academic research, and commercial products related to image processing and irregular part analysis. Next, we analyze and categorize various image processing methods used for irregular part detection. This analysis includes the study of algorithms, computer vision techniques, machine learning, and other relevant techniques that can be applied in the context of finding irregular parts in images.

After analyzing the image processing techniques, the most appropriate approach for implementing the irregular part detection module is selected. This choice is made based on the criteria of efficiency, accuracy and speed, as well as taking into account the requirements of technical development tools and the capabilities of the C++ programming language.

INFORMATION SYSTEM, DEVELOPMENT METHOD, COMPUTER VISION, IRREGULAR PARTS.

ЗМІСТ

ВСТУП

У сучасному світі зображення стають все більш важливим елементом різних сфер людської діяльності. Завдання обробки зображень включає пошук нерегулярних частин на зображеннях, що може бути дуже складним завданням, особливо при роботі з великими кількостями пікселів.

У даному дипломному проекті ми зосереджуємось на розробці модуля інформаційної системи для ефективного пошуку нерегулярних частин на зображеннях. Для досягнення цієї мети використовується сучасні методи машинного навчання та штучного інтелекту, що дозволяє покращити якість обробки зображень та робити цей процес більш швидким та точним.

Результати цього проекту можуть бути використані у різних сферах, включаючи медицину, промисловість, науку та багато інших. Наприклад, в медицині можна використовувати розроблений модуль для аналізу зображень рентгенівських знімків та виявлення патологій, а в промисловості - для контролю якості продукції. Результати дослідження можуть бути корисними також для наукових досліджень в галузі комп'ютерного зору та обробки зображень.

Однією з основних проблем пошуку нерегулярних частин на зображеннях є складність цієї задачі, особливо коли зображення має велику кількість пікселів. Це пов'язано з тим, що нерегулярні частини можуть мати різний вигляд, форму та розміри, і можуть бути розташовані в різних місцях на зображенні. Крім того, такі частини можуть бути перекриті іншими об'єктами, що робить пошук ще більш складним. Тому, розробка ефективного алгоритму для пошуку нерегулярних частин на зображенні є важливим завданням, яке потребує використання сучасних методів машинного навчання та штучного інтелекту.

Метою дипломної роботи є розробка модуля інформаційної системи для ефективного пошуку нерегулярних частин на зображеннях. Проект зосереджується на розробці алгоритмів обробки зображень та їхньої реалізації з використанням сучасних методів машинного навчання та штучного інтелекту.

1. МЕТОДИ ДЛЯ ПОШУКУ НЕРЕГУЛЯРНИХ ЧАСТИН НА ЗОБРАЖЕННЯХ

1.1 Детектори країв

Ефективна обробка зображень вимагає застосування сучасних комп'ютерних методів та технологій. Найбільш перспективними методами такої обробки є методи, засновані на застосуванні інтелектуальних технологій. Використання інтелектуальних методів та моделей в системах обробки зображень (насамперед, нейромережевих моделей), дозволяє вирішувати чимало практичних завдань, що стосуються зокрема поліпшення якості зображень та виялення нерегулярних обєктів на зображенннях.

Попереднє підвищення якості і аналізу фрагментів зображень може бути здійснено з використанням методів фільтрації та детектування кордонів частин зображень.

На сьогодні відсутній єдиний підхід до вирішення проблеми обробки зображень в рамках створення модулів спеціалізованих інформаційних систем. Методи такої обробки зображень можуть суттєво відрізнятися в залежності від того, яким шляхом зображення було отримано. Зображення, отримані за допомогою оцифрування аналогових знімків, зазвичай, викривлені шумами різних типів, серед яких можна виділити: білий гаусівський шум, що виникає при незадовільних умовах моніторингу; імпульсний шум у вигляді випадкових ізольовані точки на зображеннях.

Пониження рівня шумів в задачах обробки зображень дозволяє покращити візуальне сприйняття отриманих зображень. Виділення контурів фрагментів зображень ґрунтується на методах, що визначають точки різкої зміни яскравості зображень або присутність інших різновидів неоднорідностей. Ця функція реалізується за допомогою так званих детекторів кордонів (edge detector), що формують набір ліній, які відповідають межам об'єктів на поверхнях.

Детектори границь дають прийнятний результат при наявності стрімких змін яскравості. Втім за наявності шумів може суттєво викривлюватися інформація про стан крайових елементів, що призводить до появи помилок, пов’язаних з пропуском реальних крайових точок або з появою помилкових точок. З точки зору реалізації це означає необхідність виконання значного обсягу обчислень. Можливим варіантом реалізації завдань обробки зашумлених зображень є застосування нейромережевих методів, що дозволяють використовувати ефективні схеми обробки зображень за допомогою штучних нейронних мереж.

Фільтрація шумів здійснюється з використанням згладжуючих, медіанних та ранжуючих фільтрів. Під час фільтрації характеристики яскравості точок зображення замінюються іншими значеннями яскравості, що в меншій мірі спотворені шумами. Просторові методи поліпшення зображень, здебільшого застосовуються до растрових двовимірних зображень. Принцип таких методів полягає в застосуванні віконних операторів (масок) до кожної точки зображення. В усереднюючих фільтрах вихідними значеннями є середні значення в околі масок фільтру. В медіанних фільтрах значення пікселів представляють собою усереднені значення точок околу.

Зазвичай для зменшення впливу шуму медіанний фільтр є більш ефективним, ніж безпосереднє усереднення, тому що викликає менші спотворення кордонів виділених об’єктів.

Аналіз методів фільтрації показує, що в разі наявності імпульсних шумів для попередньої обробки зображень найбільш підходить медіанний фільтр, який добре зберігає кордони фрагментів і має високу швидкодію.

Для розпізнавання або для стиснення зображень з білим гаусовським шумом часто застосовують метод головних компонент, заснований на знаходженні базисних векторів аналізованої зони зображення. Згідно з цим методом всі зображення розбиваються на блоки, які обробляються незалежно і мають невеликі перекриття, що дозволяють запобігти ефекту блоковості під час їх стикування. Рівень деталізації характеристик сегментації залежить від конкретних завдань. Зазвичай вибираються кілька методів сегментації, які пристосовуються під

специфічні умови конкретної задачі. Сегментація в загальному випадку дві задачі: поділ зображення на фрагменти з метою виконання подальшого локального аналізу та зміна форми опису елементів зображення. Методи сегментації цифрових зображень базуються на аналізі різниці яскравостей елементів і фону зображень. При цьому обчислюються похідні у вигляді дискретних наближень градієнту.

Детектори країв (границь) є алгоритмами обробки зображень, які використовуються для виявлення границь об'єктів на зображенні. Три з найбільш відомих детекторів країв – це оператор Собеля, оператор Прюітт та оператор Кенні.

Оператор Собеля [23] – це один з найбільш широко використовуваних детекторів країв. Він працює, обчислюючи градієнти яскравості зображення в різних напрямках. Оператор Собеля використовує два ядра (матриці), що складаються з чисел, які змінюються залежно від напрямку пошуку країв. Для кожного пікселя зображення виконується згортка зображення з цими ядрами, що дає в результаті градієнти яскравості в горизонтальному та вертикальному напрямках. Градієнти потім комбінуються, щоб отримати значення градієнту яскравості та орієнтацію краю. Приклад роботи наведено на рис. 1.1.

Оператор Прюітт [22] – це детектор країв, що використовує першу похідну функції яскравості зображення. Він знаходить краї, обчислюючи зміну яскравості між сусідніми пікселями. Оператор Прюітт використовує ядро, що містить в собі числа, що змінюються залежно від напрямку краю, а потім виконує згортку зображення з цим ядром. Результатом є зображення, де кожен піксель містить значення відображення інтенсивності зміни яскравості відносно сусідніх пікселів. Приклад роботи наведено на рис. 1.2.

Оператор Кенні [21] (рис. 1.3) є одним з найбільш точних детекторів країв, який дозволяє виявляти краї з високою точністю та низьким рівнем шуму. Його робота включає:

- згладжування зображення з використанням фільтру Гаусса, що дозволяє зменшити рівень шуму та покращити точність;

- обчислення градієнту яскравості (Собеля) використовують для обчислення градієнту яскравості в горизонтальному та вертикальному напрямках;

- виявлення місць зі змінною яскравістю, відсікаються пікселі з низьким значенням градієнту;

- виявлення країв включає метод Хафа (Hough) який визначає частини ліній.

Оператор Кенні може бути досить чутливим до параметрів, таких як порогове значення, що визначає мінімальний градієнт, який буде вважатися краєм, або розмір фільтру Гаусса, що використовується для згладжування зображення. Неправильно налаштовані параметри можуть привести до великої кількості знайдених границь або великої кількості пропущених границь.

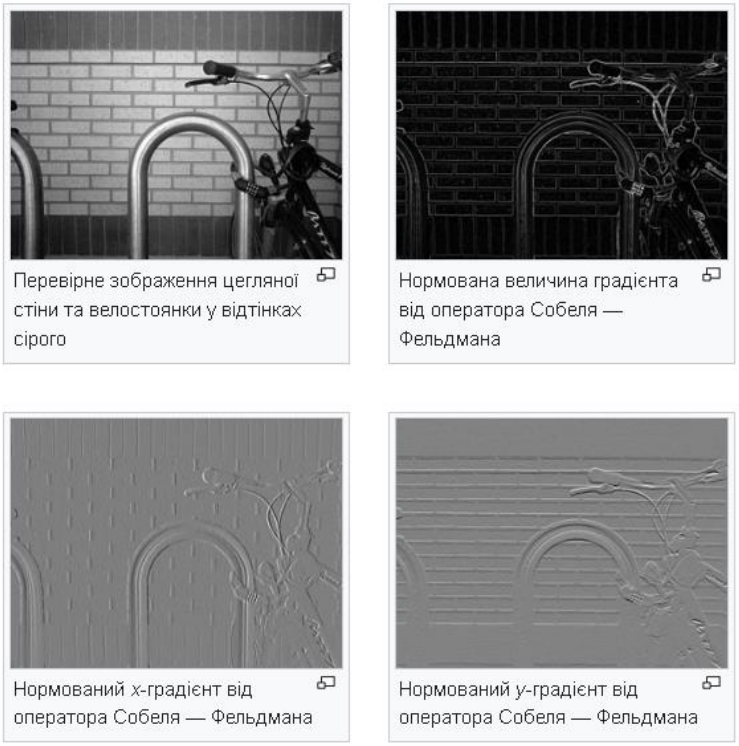

Рисунок 1.1 — Приклад роботи оператора Собеля [22]

Рисунок 1.2 — Приклад роботи оператора Прюітта [21]

Рисунок 1.3 — Приклад роботи оператора Кенні [23]

1.2 Постановка задачі

Головною задачею дипломного проєктування є розробка програмний модуля у вигляді застосунку, який використовує вбудовані методи для пошуку нерегулярних частин на зображеннях. Необхідно виконати імплементацію необхідних методів та алгоритмів, операції обробки зображень та інтерфейс користувача.

Наступною задачею є тестування розробленого модуля на різних наборах зображень з відомими нерегулярними частинами. Оцінити точність, чутливість та ефективність модуля. Треба також виконати аналіз отриманих результатів, порівняти їх з існуючими методами. Сформулювати висновки щодо ефективності та можливостей застосування розробленого модуля для пошуку нерегулярних частин зображень.

2. АНАЛІЗ МОДУЛІВ ОБРОБКИ НЕРЕГУЛЯРНИХ ЧАСТИН НА ЗОБРАЖЕННЯХ

У цьому розділі пояснювальної записки наведено аналіз окремих модулів обробки нерегулярних частин на зображеннях. Оскільки цей аналіз стосується обробки зображень, передбачається використання різних методів та алгоритмів, що дозволяють знаходити та виділяти нерегулярні частини на зображеннях. Зокрема, висвітлюються такі методи як кластерний аналіз, використання геометричних форм та виокремлення особливих точок.

Кластерний аналіз є одним з популярних методів для пошуку нерегулярних частин на зображеннях. Він використовується для групування пікселів або областей зображення в кластери на основі схожості їх характеристик. Наприклад, кластерний аналіз може бути застосований для виділення областей зі схожими текстурними ознаками або колірними характеристиками. Отримані кластери можуть служити основою для подальшого виявлення нерегулярних частин, таких як об'єкти або області зі зміненими характеристиками.

Приклад методу: K-середніх алгоритм.

Застосування: Використовується для кластеризації пікселів зображення за кольоровими характеристиками, щоб виділити різні області або об'єкти з різними кольорами. Наприклад, в дослідженнях з медичних зображень може використовуватись кластерний аналіз для виділення областей з пухлинами або аномаліями. Приклад наведено на рис. 2.1.

Іншим підходом є використання геометричних форм. Наприклад, можуть бути використані методи, такі як виявлення країв, контурів або детекторів форм. Ці методи дозволяють виділяти області на зображенні, які мають нерегулярну форму або виокремлені деталі, які відрізняються від навколишнього фону. Геометричні форми можуть бути використані для виявлення нерегулярних об'єктів, контурів або форм, що мають особливу важливість у деяких задачах, наприклад, у виявленні ракових пухлин на медичних зображеннях.

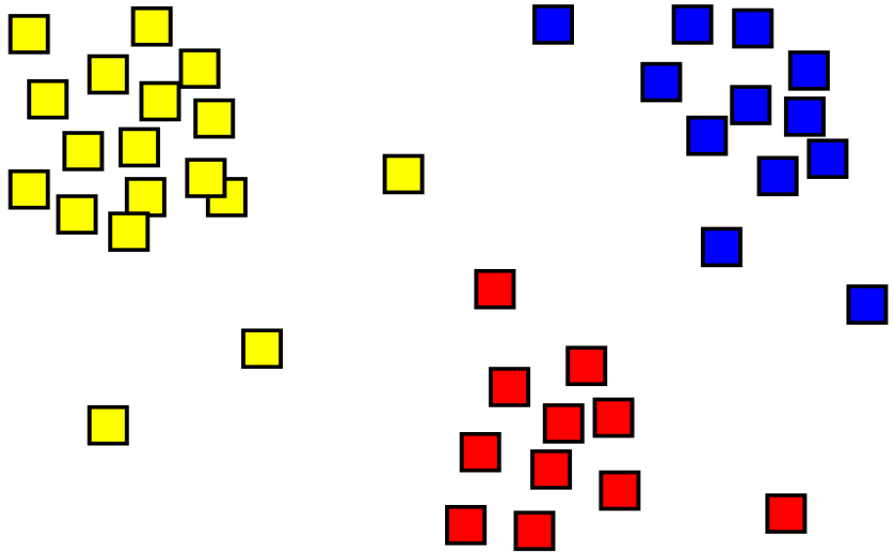

Рисунок 2.1 — Приклад методу K-середніх алгоритм [25]

Виокремлення особливих точок зосереджується на виділенні характерних пікселів на зображенні, які можуть вказувати на нерегулярні частини. Особливі точки можуть бути точками зі зміненою яскравістю, кутами, текстурою або іншими характеристиками, що відрізняються від навколишнього фону. Для виокремлення особливих точок можуть бути використані такі методя як SIFT (Scale-Invariant Feature Transform) або SURF (Speeded-Up Robust Features), які дозволяють знайти стабільні особливі точки, які є стійкими до змін масштабу, орієнтації та освітлення.

Приклад методу: SIFT (Scale-Invariant Feature Transform).

Застосування: SIFT використовується для виділення ключових точок на зображенні, які є стійкими до змін у масштабі, орієнтації та освітленні. Це може бути корисним для виявлення нерегулярних об'єктів або точок інтересу, наприклад, в розпізнаванні об'єктів на зображеннях або в стеженні за об'єктами у відеопотоці. Принцип роботи зображено на рис. 2.3.

Після виявлення масштабопросторових екстремумів (їхні розташування показано на верхньому зображенні) метод SIFT відкидає низькоконтрастні ключові точки (решту точок показано на середньому зображенні), а потім відфільтровує розташовані на контурах. Отриманий набір ключових точок показано на крайньому зображенні.

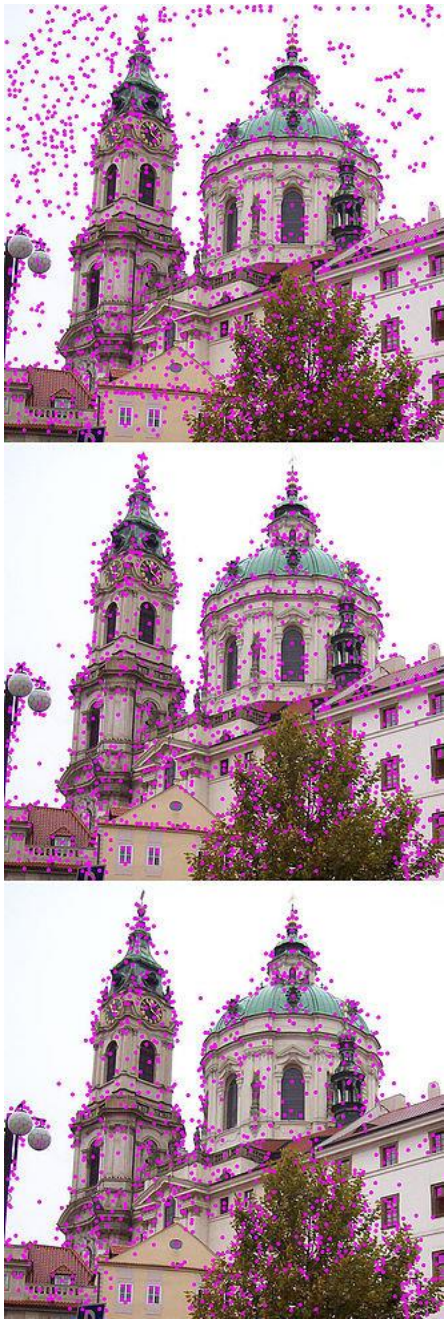

Рисунок 2.2 — Приклад роботи методу SIFT [24]

Ці приклади демонструють, як різні методи обробки зображень можуть бути застосовані для пошуку нерегулярних частин на зображеннях. Вибір конкретного методу залежить від конкретної задачі та характеристик зображень, які потрібно аналізувати.

3. РОЗРОБЛЕННЯ МОДУЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПОШУКУ НЕРЕГУЛЯРНИХ ЧАСТИН НА ЗОБРАЖЕННЯХ.

3.1 Обґрунтування вибору технічних засобів розробки

Обґрунтування вибору технічних засобів розробки модуля для пошуку нерегулярних частин на зображеннях включає нижченаведені фактори.

Швидкодія та ефективність. Мова C++ відома своєю високою швидкодією та ефективністю використання пам’яті. Це важливо для обробки зображень, оскільки зображення можуть мати великі розміри та потребують інтенсивної обчислювальної роботи. Використання мови C++ дозволяє оптимізувати обробку зображень та досягти високої продуктивності.

Наявність бібліотек обробки зображень. Мова C++ має багато потужних бібліотек для обробки зображень, таких як OpenCV, CImg, ITK та багато інших. Вони надають широкий спектр функцій та алгоритмів для обробки зображень, включаючи методи пошуку нерегулярних частин. Використання мови C++ дозволяє зручно і ефективно використовувати ці бібліотеки та їх функціонал.

Розширюваність та модульність. Мова C++ дозволяє розробляти модульні програми, де кожен компонент може бути реалізований окремо і легко замінений або розширений. Це важливо для розробки модулів обробки зображень, оскільки можуть бути потреби у використанні різних алгоритмів та методів пошуку нерегулярних частин. Мова C++ дозволяє створити модульну архітектуру, яка спрощує розширення та зміну функціоналу.

Кросплатформеність. Мова C++ є кросплатформенною і підтримується на різних операційних системах. Це означає, що модуль, розроблений за допомогою мови C++, може бути використаний на різних платформах без значних змін. Це забезпечує гнучкість та доступність модуля обробки зображень на різних пристроях і операційних системах.

Загалом, вибір мови C++ для розробки модуля пошуку нерегулярних частин на зображеннях обґрунтовується її швидкодією, наявністю потужних бібліотек

обробки зображень, розширюваністю, модульністю та кросплатформеністю. Всі ці фактори сприяють ефективній та гнучкій розробці модуля обробки зображень з можливістю пошуку нерегулярних частин.

3.2 Розроблення інтерфейсу модуля для застосунку

Розроблення інтерфейсу модуля для застосунку, який включає пошук нерегулярних частин на зображеннях, є важливою частиною проекту. Першим етапом розроблення інтерфейсу є визначення вимог до функціоналу та інтерфейсу.

В даному дипломному проєкті було реалізовано наступний функціонал:

- завантаження зображень: передбачено можливість завантажити зображення з різних джерел, таких як локальний комп'ютер, цифрова камера або сканер;

- корекція зображення: надає інструменти для корекції зображення, включаючи налаштування яскравості, контрастності, насиченості, відтінків тощо. Користувач може виправляти помилки експозиції, виправляти кольорові відхилення та вдосконалювати загальний вигляд зображення;

- ретушування: інструменти для ретушування зображень включають видалення непотрібних об'єктів або дефектів, реставрацію пошкоджених частин зображення, розмиття або розшарування ефектів, накладання фільтрів та текстур;

- обрізка та масштабування: користувач може обрізати зображення для видалення непотрібних частин або вибору певної області. Також є можливість масштабування зображення для зміни його розмірів без втрати якості;

- пошук нерегулярних частин на зображені: користувач може в ручному режимі виділити нерегулярні частини на зображенні для видалення непотрібних частин або зміни певної області;

- виправлення перспективи: корисно для фотографій будівель або пейзажів. Користувач може вирівнювати горизонт, виправляти криві лінії та надавати зображенню більш пропорційний вигляд;

- ефекти та фільтри в обробці: користувач може застосовувати різні ефекти та фільтри до зображення, включаючи чорно-білі ефекти, сепію, винтажні ефекти, розмиття, розсіяння світла та багато інших;

- збереження та експорт: після редагування зображення користувач може зберегти його у різних форматах, таких як JPEG, PNG, TIFF тощо, або експортувати в соціальні мережі або спільним доступом для інших користувачів.

На рис. 3.1 – 3.2 наведено приклад створення функції завантаження зображень.

```cpp
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для завантаження зображення
cv::Mat loadImage(const std::string& imagePath) {
    cv::Mat image;

    // Завантаження зображення з використанням OpenCV
    image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath <<
    }

    return image;
}
```

Рисунок 3.1 — Завантаження зображення в інтерфейсі з використанням мови C++ та бібліотеки OpenCV

Функція loadImage приймає шлях до зображення (imagePath) та повертає зображення у форматі cv::Mat. Функція використовує функцію cv::imread з бібліотеки OpenCV для завантаження зображення з вказаного шляху. Якщо завантаження не вдалося, функція виведе повідомлення про помилку.

Після завантаження зображення використовується функція корекції зображення в інтерфейсі з використанням мови C++ та бібліотеки OpenCV. Приклад наведено на рис 3.3 – 3.5.

```cpp
int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Виклик функції завантаження зображення
    cv::Mat loadedImage = loadImage(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (!loadedImage.empty()) {
        // Зображення успішно завантажено, можна продовжити обробку
        // Наприклад, вивести розміри зображення
        std::cout << "Розміри зображення: " << loadedImage.cols << "x" <<

        // Додатковий код для обробки зображення
    }

    return 0;
}
```

Рисунок 3.2 — Функція завантаження зображення

```cpp
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для корекції зображення
cv::Mat correctImage(const cv::Mat& image) {
    cv::Mat correctedImage;

    // Виконання корекції зображення

    // Приклад: Застосування автокорекції за допомогою функції equalizeHist(
    if (image.channels() == 1) {
        cv::equalizeHist(image, correctedImage);
    }
    else if (image.channels() == 3) {
        std::vector<cv::Mat> channels;
        cv::split(image, channels);
        for (int i = 0; i < 3; i++) {
            cv::equalizeHist(channels[i], channels[i]);
        }
        cv::merge(channels, correctedImage);
    }
```

Рисунок 3.3 — Функція корекції зображення

```
    else {
        std::cerr << "Непідтримуваний формат зображення." << std::endl;
        correctedImage = image;
    }

    return correctedImage;
}

int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath <<
        return 1;
    }
```

Рисунок 3.4 — Функція корекції зображення

```
    // Виклик функції корекції зображення
    cv::Mat correctedImage = correctImage(image);

    // Показ корегованого зображення
    cv::imshow("Corrected Image", correctedImage);
    cv::waitKey(0);

    return 0;
}
```

Рисунок 3.5 — Функція корекції зображення

Функція correctImage приймає зображення у форматі cv::Mat (image) та повертає скориговане зображення у тому ж форматі. Функція виконує корекцію зображення, наприклад, застосовує автокорекцію для поліпшення яскравості та контрастності зображення за допомогою функції equalizeHist() з бібліотеки

OpenCV. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про помилку. Потім викликається функція корекції correctImage для обробки зображення. Отримане скориговане зображення відображається за допомогою функції cv::imshow. В процесі використовується функція ретушування. Приклад наведено на рис 3.6 – 3.7.

```
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для ретушування зображення
cv::Mat retouchImage(const cv::Mat& image) {
    cv::Mat retouchedImage;

    // Виконання ретушування зображення

    // Приклад: Застосування фільтру розмиття для зменшення дрібних дефектів
    cv::GaussianBlur(image, retouchedImage, cv::Size(5, 5), 0);

    return retouchedImage;
}
```

Рисунок 3.6 — Функція ретушування зображення

```
int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath <<
        return 1;
    }

    // Виклик функції ретушування зображення
    cv::Mat retouchedImage = retouchImage(image);

    // Показ ретушованого зображення
    cv::imshow("Retouched Image", retouchedImage);
    cv::waitKey(0);

    return 0;
}
```

Рисунок 3.7 — Функція ретушування зображення

Функція retouchImage приймає зображення у форматі cv::Mat (image) та повертає ретушоване зображення у тому ж форматі. Функція виконує ретушування зображення, наприклад, застосовує фільтр розмиття (cv::GaussianBlur) для зменшення дрібних дефектів на зображенні. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про помилку. Потім викликається функція ретушування retouchImage для обробки зображення. Отримане ретушоване зображення відображається за допомогою функції cv::imshow.

Потім функція для ідеалізаціїї зображення під цілі користувача, обрізка та масштабування. Це наведено на рис. 3.8 – 3.10.

```cpp
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для обрізки та масштабування зображення
cv::Mat cropAndResizeImage(const cv::Mat& image, int x, int y, int width
    cv::Rect roi(x, y, width, height);
    cv::Mat croppedImage = image(roi);
    cv::Mat resizedImage;

    // Масштабування зображення до нових розмірів
    cv::resize(croppedImage, resizedImage, cv::Size(newWidth, newHeight)

    return resizedImage;
}
```

Рисунок 3.8 — Функція обрізки та масштабування зображення

```cpp
int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath <<
        return 1;
    }

    int x = 100; // Початкова координата X для обрізки
    int y = 100; // Початкова координата Y для обрізки
    int width = 300; // Ширина для обрізки
    int height = 200; // Висота для обрізки
    int newWidth = 200; // Нова ширина після масштабування
    int newHeight = 150; // Нова висота після масштабування

    // Виклик функції обрізки та масштабування зображення
    cv::Mat croppedResizedImage = cropAndResizeImage(image, x, y, width,
```

Рисунок. 3.9 — Функція обрізка та масштабування зображення

```
    // Показ обрізаного та масштабованого зображення
    cv::imshow("Cropped and Resized Image", croppedResizedImage);
    cv::waitKey(0);

    return 0;
}
```

Рисунок 3.10 — Функція обрізки та масштабування зображення

Функція cropAndResizeImage приймає зображення у форматі cv::Mat (image) та виконує обрізку та масштабування зображення. Вона отримує параметри, такі як початкові координати (x, y), ширина та висота обрізки (width, height), а також нові розміри після масштабування (newWidth, newHeight). Функція обрізає зображення з використанням регіону інтересу (cv::Rect) і потім масштабує його до нових розмірів за допомогою функції cv::resize. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про помилку. Потім викликається функція обрізки та масштабування cropAndResizeImage з встановленими параметрами. Отримане обрізане та масштабоване зображення відображається за допомогою функції cv::imshow.

В результаті відбувається перехід до основної функції пошуку нерегулярних частин на зображені, що зображено на рис. 3.11 – 3.13.

Функція findIrregularParts приймає зображення у форматі cv::Mat (image) та виконує пошук нерегулярних частин на зображенні. Спочатку зображення перетворюється в чорно-біле шляхом конвертації в відтінки сірого за допомогою функції cv::cvtColor. Потім виконується порогова обробка для отримання бінарного зображення з використанням функції cv::threshold. Далі можна застосовувати різні алгоритми для аналізу та обробки нерегулярних частин на зображенні. У прикладі показано використання алгоритму знайдення контурів за допомогою функції cv::findContours. Отримані контури потім відображаються на оригінальному зображенні з використанням функції cv::drawContours. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім

перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про помилку. Потім викликається функція пошуку нерегулярних частин findIrregularParts з завантаженим зображенням як параметром. Знайдені нерегулярні частини відображаються у вікні за допомогою функції cv::imshow.

```
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для пошуку нерегулярних частин на зображенні
void findIrregularParts(const cv::Mat& image) {
    cv::Mat grayImage;
    cv::cvtColor(image, grayImage, cv::COLOR_BGR2GRAY);

    cv::Mat binaryImage;
    cv::threshold(grayImage, binaryImage, 0, 255, cv::THRESH_BINARY_INV

    // Виконання аналізу та обробки нерегулярних частин

    // Приклад: Застосування алгоритму знайдення контурів
    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(binaryImage, contours, cv::RETR_EXTERNAL, cv::CHAIN_
```

Рисунок 3.11 — Функція пошуку нерегулярних частин на зображені

```
    // Відображення результатів
    cv::Mat resultImage = image.clone();
    cv::drawContours(resultImage, contours, -1, cv::Scalar(0, 0, 255), 2

    // Показ зображення з виділеними нерегулярними частинами
    cv::imshow("Irregular Parts", resultImage);
    cv::waitKey(0);
}

int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath <<
        return 1;
    }
```

Рисунок 3.12 — Функція пошуку нерегулярних частин на зображені

```
    // Виклик функції пошуку нерегулярних частин
    findIrregularParts(image);

    return 0;
}
```

Рисунок 3.13 — Функція пошуку нерегулярних частин на зображені

Існує також можливість використати функцію виправлення перспективи. Приклад її використання наведено на рис. 3.14 – 3.17.

```
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для виправлення перспективи зображення
cv::Mat correctPerspective(const cv::Mat& image, const std::vector<cv::Point2f
    >& corners) {
    cv::Point2f topLeft(0, 0);
    cv::Point2f topRight(image.cols - 1, 0);
    cv::Point2f bottomLeft(0, image.rows - 1);
    cv::Point2f bottomRight(image.cols - 1, image.rows - 1);

    // Визначення нових розмірів зображення
    cv::Size targetSize(image.cols, image.rows);
```

Рисунок 3.14 — Функція виправлення перспективи

```
    // Створення матриці перспективної трансформації
    cv::Mat perspectiveTransform =
cv::getPerspectiveTransform(corners, {topLeft,
topRight, bottomLeft, bottomRight});

    // Виправлення перспективи
    cv::Mat correctedImage;
    cv::warpPerspective(image, correctedImage,
perspectiveTransform, targetSize);

    return correctedImage;
}
```

Рисунок 3.15 — Функція виправлення перспективи

```
int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath << std::endl;
        return 1;
    }

    // Визначення кутиків перспективи (верхній лівий, верхній правий, нижній лівий, нижній
правий)
    std::vector<cv::Point2f> corners;
    corners.push_back(cv::Point2f(100, 100)); // Верхній лівий кут
    corners.push_back(cv::Point2f(500, 100)); // Верхній правий кут
    corners.push_back(cv::Point2f(100, 400)); // Нижній лівий кут
    corners.push_back(cv::Point2f(500, 400)); // Нижній правий кут
```

Рисунок 3.16 — Функція виправлення перспективи

```
    // Виклик функції виправлення перспективи
    cv::Mat correctedImage = correctPerspective(image, corners);

    // Показ виправленого зображення
    cv::imshow("Corrected Image", correctedImage);
    cv::waitKey(0);

    return 0;
}
```

Рисунок 3.17 — Функція виправлення перспективи

У цьому прикладі функція correctPerspective приймає зображення у форматі cv::Mat (image) та вектор точок (corners), які визначають кути перспективи, які потрібно виправити. Функція створює матрицю перспективної трансформації за допомогою функції cv::getPerspectiveTransform з початковими та цільовими точками та використовує її для виправлення перспективи зображення за допомогою функції cv::warpPerspective. Результат виправленого зображення повертається як вихідний параметр. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про

помилку. Потім визначаються кути перспективи, які потрібно виправити, та викликається функція виправлення перспективи correctPerspective з завантаженим зображенням та вектором точок кутів. Отримане виправлене зображення відображається у вікні за допомогою функції cv::imshow. Цей код можна розширити та адаптувати залежно від потреб і специфіки виправлення перспективи зображення. Приклад наведено на рис. 3.18 – 3.20.

```
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для додавання фільтрів та ефектів до зображення
cv::Mat applyFilters(const cv::Mat& image) {
    cv::Mat filteredImage;

    // Приклад: Застосування фільтру Гаусса
    cv::GaussianBlur(image, filteredImage, cv::Size(5, 5), 0);

    // Приклад: Застосування ефекту сепії
    cv::Mat sepiaImage = image.clone();
    for (int y = 0; y < sepiaImage.rows; ++y) {
        for (int x = 0; x < sepiaImage.cols; ++x) {
            cv::Vec3b& pixel = sepiaImage.at<cv::Vec3b>(y, x);
            pixel[0] = static_cast<uchar>(0.272 * pixel[2] + 0.534 * pixel[1] + 0.131 *
pixel[0]);
            pixel[1] = static_cast<uchar>(0.349 * pixel[2] + 0.686 * pixel[1] + 0.168 *
pixel[0]);
            pixel[2] = static_cast<uchar>(0.393 * pixel[2] + 0.769 * pixel[1] + 0.189 *
pixel[0]);
        }
```

Рисунок 3.18 — Функція обробки ефектами і фільтрами

```
    }

    // Повернення обробленого зображення
    return sepiaImage;
}

int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath << std::endl;
        return 1;
    }

    // Виклик функції для застосування фільтрів та ефектів
    cv::Mat processedImage = applyFilters(image);

    // Показ обробленого зображення
    cv::imshow("Processed Image", processedImage);
    cv::waitKey(0);

    return 0;
```

Рисунок 3.19 — Функція обробки ефектами і фільтрами

```
    }

    // Повернення обробленого зображення
    return sepiaImage;
}

int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath << std::endl;
        return 1;
    }

    // Виклик функції для застосування фільтрів та ефектів
    cv::Mat processedImage = applyFilters(image);

    // Показ обробленого зображення
    cv::imshow("Processed Image", processedImage);
    cv::waitKey(0);

    return 0;
```

Рисунок 3.20 — Функція обробки ефектами і фільтрами

У цьому прикладі функція applyFilters приймає зображення у форматі cv::Mat (image) та застосовує до нього фільтри та ефекти. У прикладі показано два приклади: застосування фільтру Гауса за допомогою функції cv::GaussianBlur та застосування ефекту сепії шляхом зміни каналів зображення. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про помилку. Потім викликається функція applyFilters з завантаженим зображенням. Отримане оброблене зображення відображається у вікні за допомогою функції cv::imshow. Цей код можна розширити та адаптувати залежно від ваших потреб і специфіки додавання фільтрів та ефектів до зображень.

В кінці використовується функція збереження та експорт зображення. Приклад наведено на рис. 3.21 – 3.23.

У цьому прикладі функція saveImage приймає зображення у форматі cv::Mat (image) та шлях до вихідного файлу (outputPath). Функція використовує функцію cv::imwrite з бібліотеки OpenCV для збереження зображення у файл. Вона повертає значення типу bool, що вказує на успішність збереження. Після збереження виводиться відповідне повідомлення. У головній функції (main) спочатку завантажується зображення за допомогою функції cv::imread. Потім перевіряється, чи вдалося завантажити зображення. Якщо завантаження не вдалося, виводиться повідомлення про помилку. Потім виконується обробка зображення. Наприклад, можна застосувати фільтри або ефекти до зображення. Нарешті, викликається функція saveImage для збереження та експорту обробленого зображення за допомогою шляху outputPath. Цей код можна розширити та адаптувати залежно від ваших потреб і специфіки збереження та експорту зображень.

```cpp
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

// Функція для збереження та експорту зображення
void saveImage(const cv::Mat& image, const std::string& outputPath) {
    bool success = cv::imwrite(outputPath, image);

    if (success) {
        std::cout << "Зображення успішно збережено у файлі: " << outputPath << std::endl;
    } else {
        std::cerr << "Помилка при збереженні зображення" << std::endl;
    }
}
```

Рисунок 3.21 — Функція збереження та експорт зображення в інтерфейсі з кодом

```cpp
int main() {
    std::string imagePath = "image.jpg"; // Шлях до зображення
    std::string outputPath = "output.jpg"; // Шлях для збереження обробленого зображення

    // Завантаження зображення з використанням OpenCV
    cv::Mat image = cv::imread(imagePath);

    // Перевірка, чи вдалося завантажити зображення
    if (image.empty()) {
        std::cerr << "Не вдалося завантажити зображення: " << imagePath << std::endl;
        return 1;
    }

```

Рисунок 3.22 — Функція збереження та експорт зображення в інтерфейсі з кодом

```cpp
    // Обробка зображення...

    // Збереження та експорт обробленого зображення
    saveImage(image, outputPath);

    return 0;
}
```

Рисунок 3.23 — Функція збереження та експорт зображення в інтерфейсі з кодом

3.3 Проєктування користувацького інтерфейсу

На основі вимог розробляється дизайн користувацького інтерфейсу. Цей етап включає розташування елементів інтерфейсу, вигляд кнопок, полів введення, списків тощо. Важливо забезпечити зрозумілість та зручність взаємодії користувача з модулем.

3.3.1 Розроблення функціональності

Після проектування інтерфейсу реалізується функціональність модуля. Цей етап включає розроблення алгоритмів пошуку нерегулярних частин на зображеннях, їх інтеграцію з інтерфейсом та забезпечення взаємодії з іншими компонентами програми. Програмний застосунок повинен відображати результати роботи. Це може бути у вигляді виділення частин зображення, маркерів або додаткових інформаційних елементів. Візуалізацію повинна бути зрозумілою та інтуїтивною для користувача.

3.3.2 Тестування та вдосконалення

Після розробки інтерфейсу проведено тестування для перевірки правильності роботи застосунку, зручності його використання та відповідності вимогам. На основі результатів тестування вносяться необхідні зміни та вдосконалення до інтерфейсу. Розроблення інтерфейсу модуля для пошуку нерегулярних частин на зображеннях вимагає уваги до деталей, зрозумілості та зручності взаємодії з користувачем. Використання мови програмування C++ дозволяє реалізувати потрібну функціональність та забезпечити швидку та ефективну роботу модуля. Для роботи програмного застосунку були використані зображення, що наведені на рис. 3.24.

Рисунок 3.24 — Приклад досліджувальних зображень

Інтерфейс містить різні функції та опції для редагування та обробки зображень. Основною метою інтерфейсу є надання користувачеві зручних інструментів для виконання різноманітних операцій зі зображеннями. Інтерфейс програми зображено на рис. 3.25.

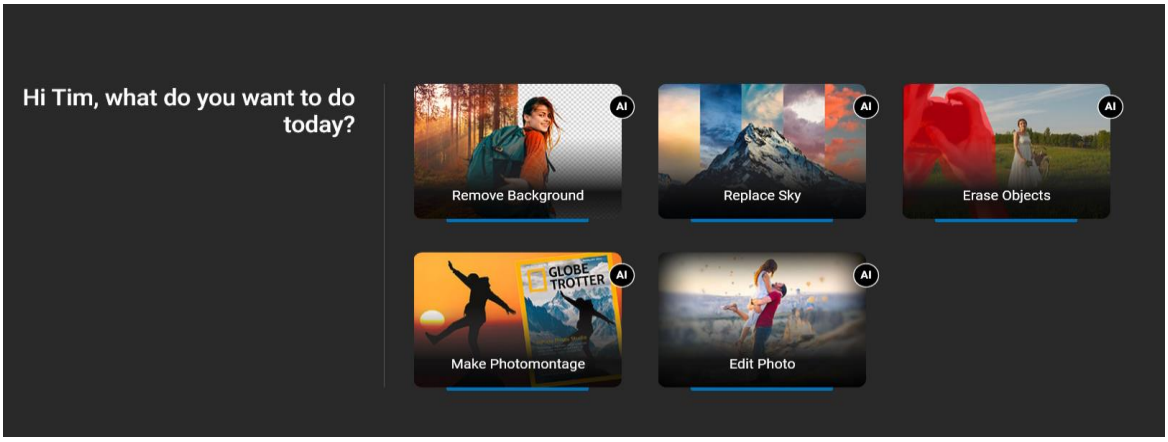

Рисунок 3.25 — Інтерфейс програми після запуску

Вибір зображення може відбуватися декількома способами, залежно від конкретної реалізації програми:

- діалоговим вікном вибору файлу – ця функція викликає діалогове вікно вибору файлу. Користувач може переглядати свої файли, використовуючи навігацію по файловій системі і вибирати бажане зображення зі списку файлів або його розташування;

- перетягування та відпускання – ця функція дає змогу перетягнути зображення з файлового провідника або іншого джерела та відпустити його в спеціальну область інтерфейсу, яка приймає зображення. Приклад наведено на рис. 3.26.

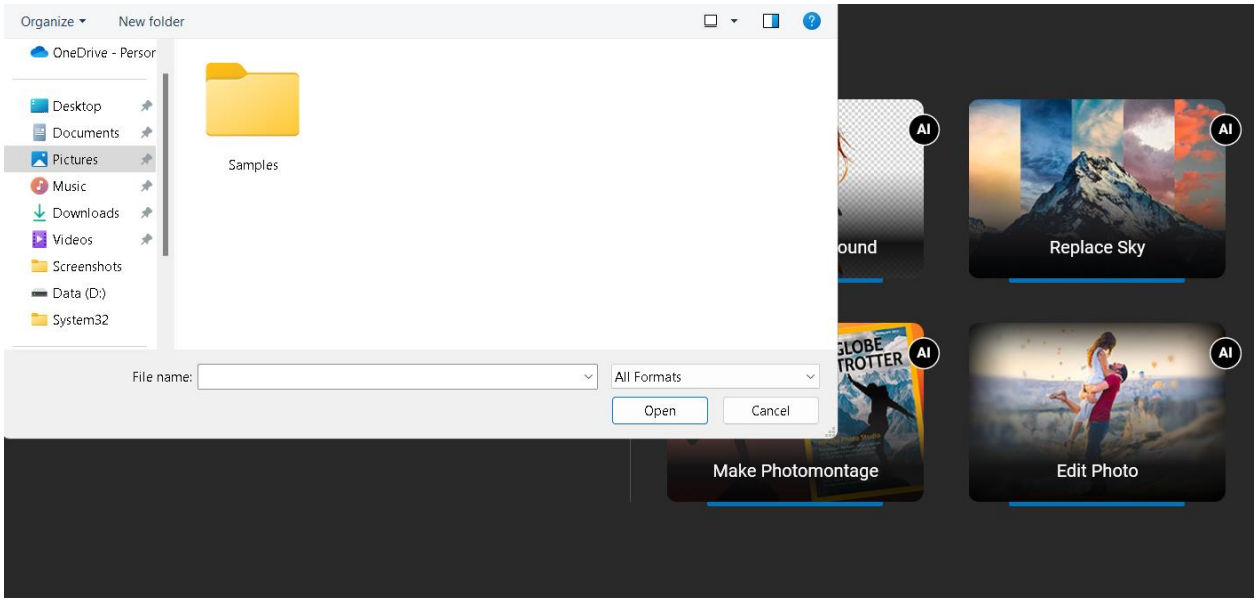

Рисунок 3.26 — Діалогове вікно вибору файлу

Користувач завантажує зображення в інтерфейс програми за допомогою вищезгаданих методів, після чого обране зображення відображається на екрані. Приклад наведено на рис. 3.27.

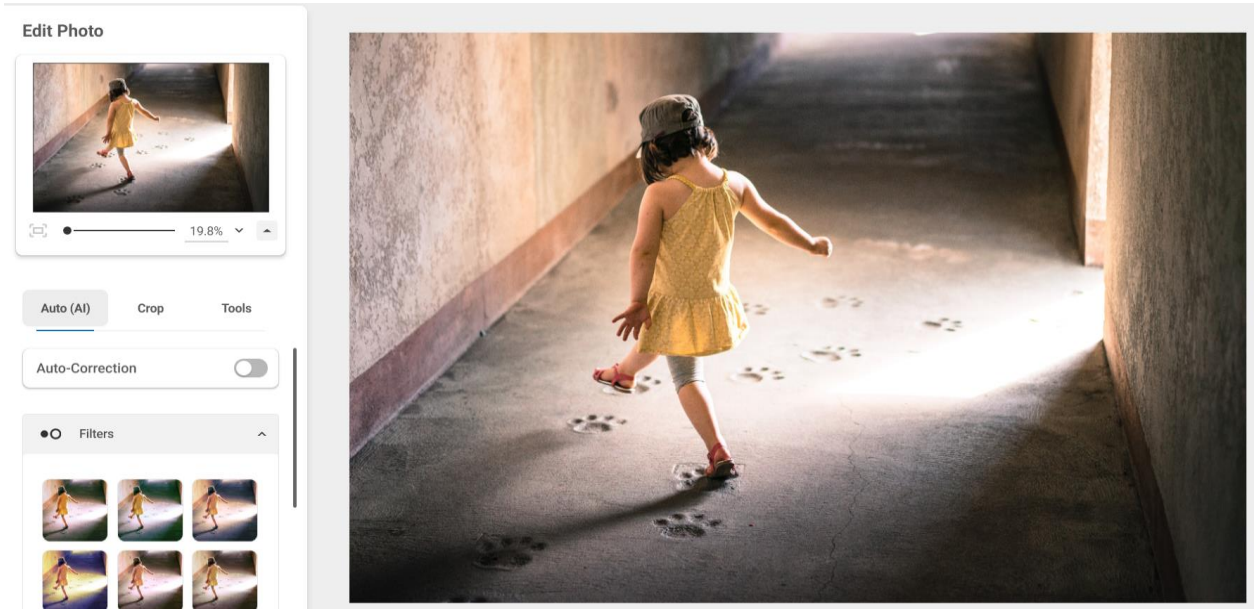

Рисунок 3.27 —Завантажене фото

3.3.3 Вибір алгоритму пошуку

Інтерфейс має кілька реалізованих алгоритмів аналізу зображень для виявлення нерегулярних частин. Користувач може обрати потрібний алгоритм або налаштувати параметри пошуку. Вибір алгоритму пошуку зображено на рис 3.28.

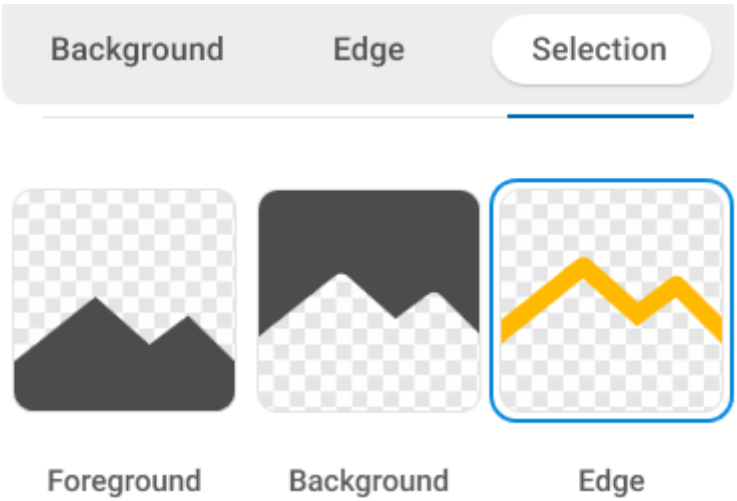

Рисунок 3.28 – Вибір алгоритмів пошуку

3.3.4 Виконання аналізу зображення

Обрані алгоритми аналізують зображення з метою виявлення нерегулярних частин. Це може включати пошук контурів, країв, текстури або інших характеристик, що відрізняються від регулярних областей.

Для роботи було обрано виділення нерегулярних частин: після аналізу зображення алгоритми виокремлюють нерегулярні частини, виявлені на зображенні. Це може бути виконано за допомогою виділення контурів, зафарбування або інших методів, що демонструють виділення цих частин на зображенні. Приклад виконання наведено на рис. 3.29 – 3.33.

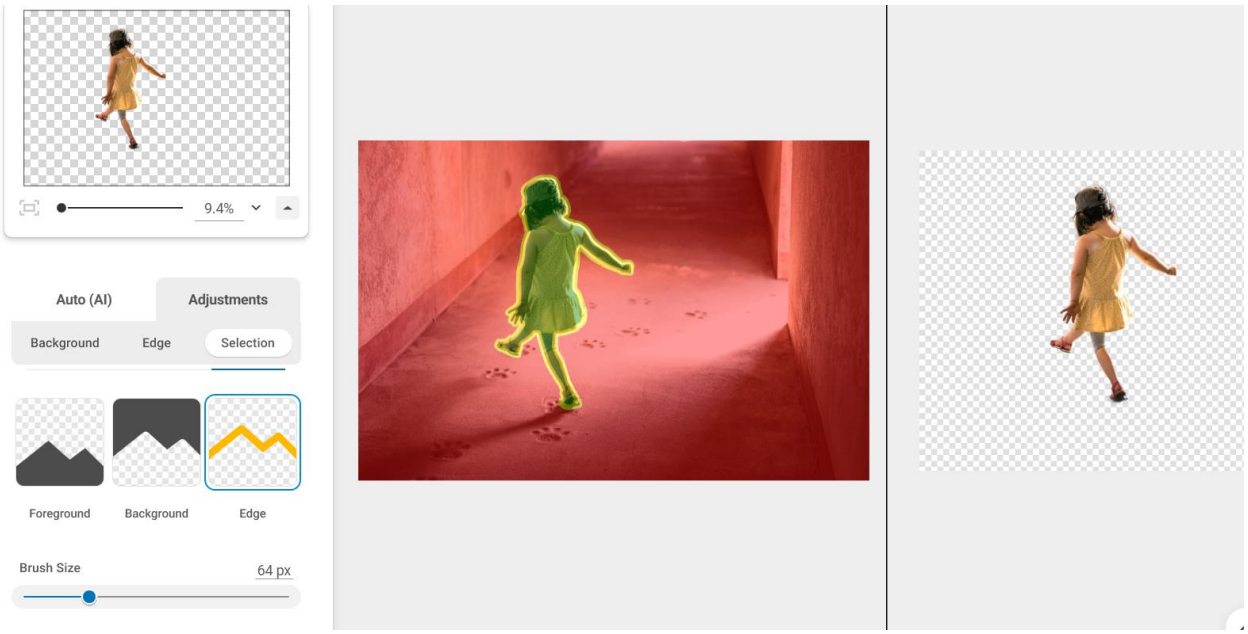

Рисунок 3.29 – Результат роботи програми для зображення 1

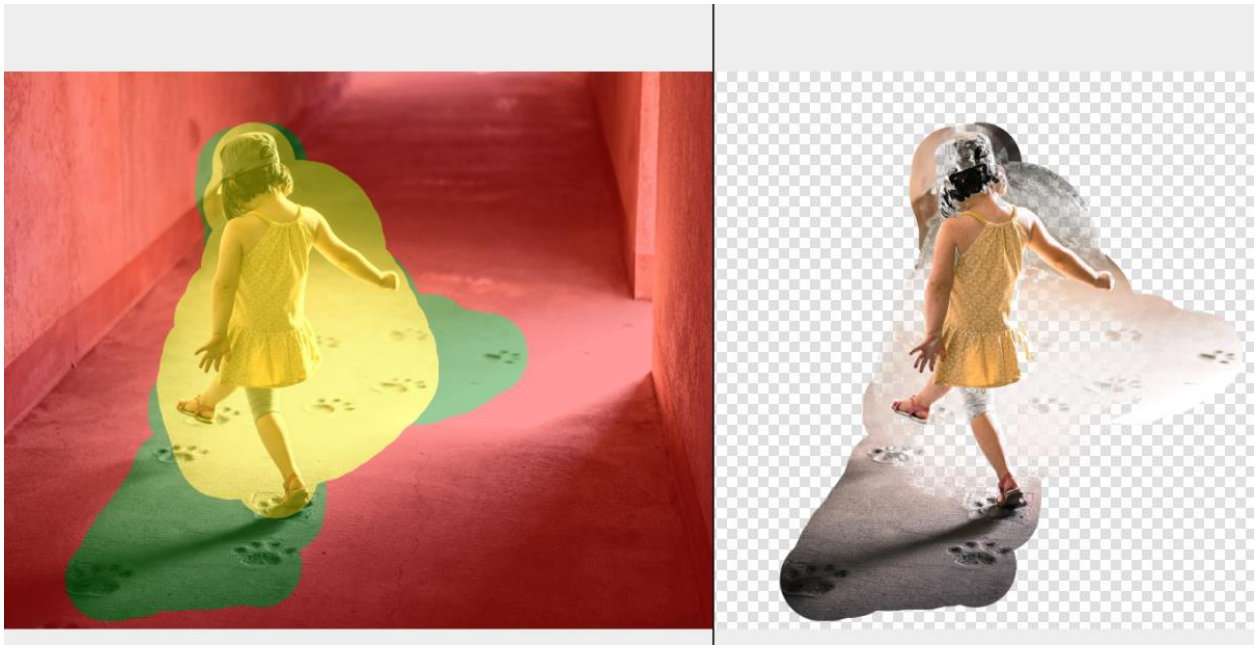

Рисунок 3.30 – Результат роботи програми для зображення 1.

Рисунок 3.31 – Результат роботи програми для зображення 1.

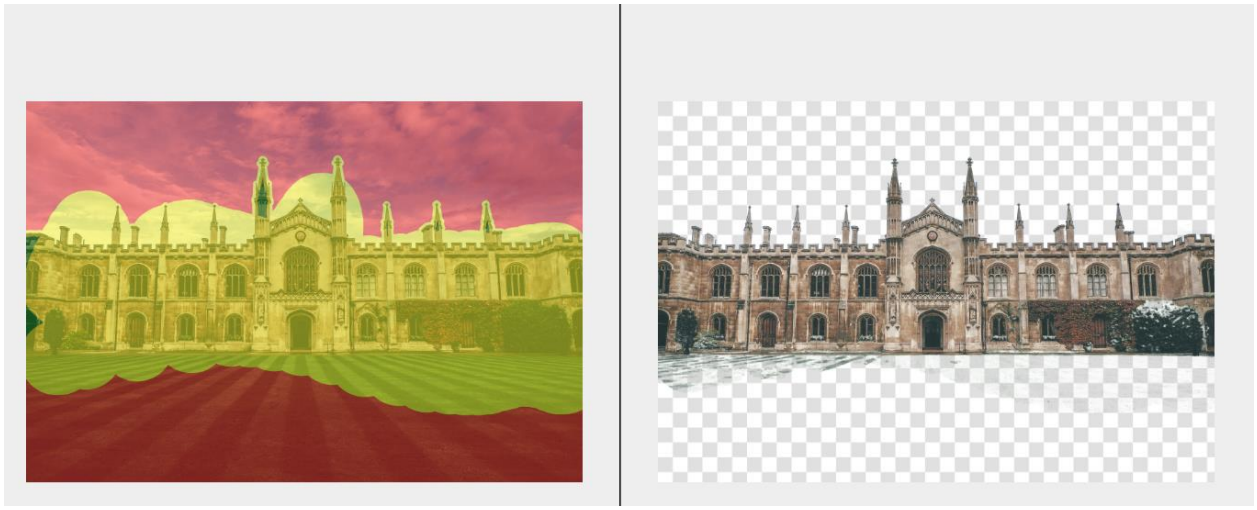

Рисунок 3.32 – Результат роботи програми для зображення 2.

Рисунок 3.33 – Результат роботи програми для зображення 2

3.3.5 Відображення результатів

Інтерфейс відображає результати аналізу зображення, показуючи виділені нерегулярні частини на зображенні. Це дозволяє користувачеві побачити та оцінити результати пошуку. Крім цього, користувач може мати можливість налаштувати параметри пошуку нерегулярних частин, такі як чутливість алгоритмів, розмір мінімальної або максимальної області, що розглядається, чи інші параметри, що впливають на точність та результати пошуку.

3.4 Аналіз результатів моделювання

Аналіз результатів моделювання може включати оцінку якості, візуальну перевірку та подальшу обробку зображень. Аспекти, які можуть бути враховані при аналізі результатів:

- якість: наскільки точно і вірно виконана модифікація зображення. Це включає перевірку наявності артефактів, затемнень, розмиття або інших небажаних змін у зображенні. Якщо обробка не задовольняє очікування, можуть бути застосовані додаткові кроки для поліпшення результатів;

- візуальна перевірка: після обробки зображення користувач може переглянути результати для оцінки візуального враження. Це включає перевірку змін у текстурі, кольорах, контрасті та інших візуальних характеристиках зображення. Візуальний аналіз дозволяє виявити нерегулярності або потребу в додатковій обробці;

- подальша обробка зображення: якщо після обробки виявляються недоліки або потреба в додаткових змінах, інтерфейс може надає інструменти для подальшої обробки зображень. Це інструменти для ретуші, корекції кольору, регулювання контрастності, застосування фільтрів, інші можливості для поліпшення зображення після модулювання;

- порівняння з оригіналом: користувач може порівняти модифіковане зображення з оригіналом для оцінки виконаних змін. Це дозволяє виявити,

наскільки успішно виконано обробку та наскільки вдало змінено вихідне зображення. Важливо зазначити, що процес аналізу результатів може бути індивідуальним для кожного користувача та їхніх конкретних потреб. Інтерфейс може надавати різноманітні інструменти та можливості для аналізу та поліпшення результатів аналізу та обробки зображень.

Також аспекти, які можуть бути враховані при аналізі результатів пошуку нерегулярних частин:

1. Точність пошуку: важливо оцінити точність пошуку нерегулярних частин на зображенні. Це включає перевірку, наскільки добре алгоритм аналізу зображення виявляє нерегулярні області та уникне хибно-позитивів або хибно-негативних результатів. Користувач може оцінити різні показники точності, такі як чутливість, специфічність, точність, F-меру тощо;

2. Візуальна перевірка: після пошуку нерегулярних частин користувач може переглянути результати для візуальної оцінки. Це включає перевірку виділених областей, контурів чи інших показників, що показують виявлені нерегулярні частини на зображенні. Візуальна перевірка дозволяє користувачу побачити, наскільки точно були виявлені та виділені нерегулярні частини;

3. Подальша обробка зображення: надаються можливості для подальшої обробки знайдених нерегулярних частин. Наприклад, користувач може застосовувати фільтри, ефекти, коригувати яскравість, контрастність, розмір тощо для поліпшення виділених областей або досягнення бажаного візуального ефекту;

4. Відображення результатів: відображаються результати пошуку нерегулярних частин, показуючи їхнє місцезнаходження, розмір, контур або інші візуальні показники. Це дозволяє користувачу краще розуміти результати пошуку та взаємодіяти з ними для подальших дій.

ВИСНОВКИ

У рамках виконання дипломного проєкту було проведено аналіз методів обробки зображень, зокрема алгоритмів та засобів пошуку нерегулярних частин. Результати досліджень та розробки підтверджують, що існують можливості для ефективного виявлення та обробки нерегулярних областей на зображеннях.

Проаналізовано кожен модуль застосунку, що розроблявся, окремо, від розроблення інтерфейсу для завантаження зображення до функціоналу, пов'язаного з пошуком нерегулярних частин, виправленням перспективи, додаванням фільтрів та ефектів, а також збереженням та експортом зображення. Кожна функція була описана з відповідним кодом для більшої ясності та розуміння.

Важливою складовою дипломної роботи був аналіз якості результатів обробки та пошуку нерегулярних частин на зображеннях. Були враховані аспекти, такі як точність пошуку, візуальна перевірка, подальша обробка та порівняння з оригіналом. Виявлено, що інтерфейс програмного застосунка дозволяє зручно аналізувати результати, вносити корективи та покращувати обробку зображень.

Отже, реалізація дипломного проєкту з пошуку нерегулярних частин на зображенні вирішує актуальну задачу обробки зображень. Розроблений інтерфейс програмного застосунку забезпечує широкий функціонал та зручність в роботі з нерегулярними частинами, дозволяючи ефективно виявляти, редагувати та обробляти їх.

Дана дипломна робота має практичне значення та може бути використана в різних сферах, таких як обробка зображень, комп'ютерне зорове сприйняття, медична діагностика, машинне навчання та багато інших.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дубінський, І. В. (2014). Комп'ютерне бачення: методи та моделі. Видавництво Львівської політехніки. URL: http://comvis.lp.edu.ua/book/ (дата звернення: 30.05.2023).

2. Митріщук, Б., & Карабахтій, Ю. (2013). Основи цифрової обробки зображень. Національний університет "Львівська політехніка". URL: http://dip.lp.edu.ua/ (дата звернення: 30.05.2023).

3. Сєров, О. П., & Коробка, І. М. (2014). Обробка та аналіз біомедичних зображень. Видавничий дім "Слово". URL: https://dspace.nbuv.gov.ua/handle/123456789/97216 (дата звернення: 30.05.2023).

4. Патерік, С. В., & Титаренко, М. Б. (2015). Методи та засоби комп'ютерного зору. Львівський національний університет імені Івана Франка. URL: http://comvis.univ.kiev.ua/textbook/ (дата звернення: 30.05.2023).

5. Шульженко, Н. В., & Салій, В. П. (2016). Математичні методи обробки зображень. Київський національний університет імені Тараса Шевченка. URL: http://www.imageprocessing.com.ua/ (дата звернення: 30.05.2023).

6. Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer. URL: http://szeliski.org/Book/ (дата звернення: 30.05.2023).

7. Forsyth, D. A., & Ponce, J. (2012). Computer Vision: A Modern Approach. Pearson. URL: http://www.computervisionmodels.com/ (дата звернення: 30.05.2023).

8. Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2009). Digital Image Processing Using MATLAB. Gatesmark Publishing. URL: https://www.imageprocessingplace.com/ (дата звернення: 30.05.2023).

9. Sonka, M., Hlavac, V., & Boyle, R. (2014). Image Processing, Analysis, and Machine Vision. Cengage Learning. URL: https://www.cvl.iis.u-tokyo.ac.jp/~kawamura/ima/vision/sonka/ (дата звернення: 30.05.2023).

10. Burger, W., & Burge, M. (2016). Digital Image Processing: An Algorithmic Introduction Using Java. Springer. URL:

https://www.sciencedirect.com/book/9781848829340/digital-image-processing (дата звернення: 30.05.2023).

11. Ma, S., & Fu, H. (2017). Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools. CRC Press. URL: https://www.crcpress.com/Digital-Image-Processing-and-Analysis-Human-and-Computer-Vision-Applications/Ma-Fu/p/book/9781498769167 (дата звернення: 30.05.2023).

12. Ballard, D. H., & Brown, C. M. (1982). Computer Vision. Prentice Hall. URL: https://mitpress.mit.edu/books/computer-vision (дата звернення: 30.05.2023).

13. Hartley, R., & Zisserman, A. (2003). Multiple View Geometry in Computer Vision. Cambridge University Press. URL: http://www.robots.ox.ac.uk/~vgg/hzbook/ (дата звернення: 30.05.2023).

14. Jain, A. K. (1989). Fundamentals of Digital Image Processing. Prentice Hall. URL: https://link.springer.com/book/10.1007/978-94-011-3122-0 (дата звернення: 30.05.2023).

15. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media. URL: https://www.oreilly.com/library/view/learning-opencv/9780596516130/ (дата звернення: 30.05.2023).

16. Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach. Pearson. URL: http://aima.cs.berkeley.edu/ (дата звернення: 30.05.2023).

17. Shi, J., & Tomasi, C. (1994). Good Features to Track. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. URL: https://ieeexplore.ieee.org/document/323794 (дата звернення: 30.05.2023).

18. Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 60(2), 91-110 URL: https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94 (дата звернення: 30.05.2023).

19. Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. Proceedings of the IEEE Conference on Computer Vision

and Pattern Recognition. URL: https://ieeexplore.ieee.org/document/990517 (дата звернення: 30.05.2023).

20. Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. URL: https://ieeexplore.ieee.org/document/1467360 (дата звернення: 30.05.2023).

21. Алгоритм Кенні. URL: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%B5%D0%BD%D0%BD%D1%96 (дата звернення: 30.05.2023).

22. Оператор Прюітт. URL: http://surl.li/hzpkk (дата звернення: 30.05.2023).

23. Оператор Собеля URL: https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%A1%D0%BE%D0%B1%D0%B5%D0%BB%D1%8F (дата звернення: 30.05.2023).

24. Масштабоінваріантне ознакове перетворення. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%BE%D1%96%D0%BD%D0%B2%D0%B0%D1%80%D1%96%D0%B0%D0%BD%D1%82%D0%BD%D0%B5_%D0%BE%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%B2%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F. (дата звернення: 30.05.2023).

25. Кластеризація методом к–середніх. URL: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BC_%D0%BA%E2%80%93%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BD%D1%96%D1%85 (дата звернення: 30.05.2023).