

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО ”

Факультет електроніки
Кафедра електронної інженерії

"На правах рукопису"
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ В.І. Тимофєєв
“ ____ ” _____ 20 ____ р.

Магістерська дисертація

зі спеціальності 153 мікро- та наносистемна техніка
на тему Аналізатор якості зображення з відеокамер

Виконав: студент 2 курсу, групи ДМ-21мн

Васільєв Максим Олегович

(прізвище, ім'я, по батькові)



(підпис)

Науковий керівник Доц., к.т.н., доц. Вунтесмері Ю. В.

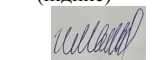
(посада, вчене звання, науковий ступінь, прізвище та ініціали)



(підпис)

Рецензент проф. каф.ЕП,д.т.н.,проф.Мельник І. В.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)



(підпис)

Засвідчую, що у цій магістерській дисертації немає
запозичень з праць інших авторів без відповідних посилань.

Студент 
(підпис)

Київ - 2024 року

Форма № Н-9.01

**Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”**

Факультет електронікиКафедра електронної інженеріїРівень вищої освіти – другий (магістерський) за освітньо-науковою програмоюСпеціальність 153 мікро- та наносистемна техніка**ЗАТВЕРДЖУЮ****Завідувач кафедри**

В.І. Тимофєєв

“ ____ ” _____ 20__ р.

З А В Д А Н Я**НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ**Васільєв Максим Олегович

(прізвище, ім'я, по батькові)

1. Тема дисертації - Аналізатор якості зображення з відеокамерНауковий керівник Доц., к.т.н., доц. Вунтесмері Ю. В.,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “01” квітня 2024 року № 1518-с

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження – процес оцінки якості зображення з камер відеоспостереження.4. Предмет дослідження – алгоритми детекції порушень якості зображення.5. Перелік питань, які потрібно розробити 1. Методи аналізу зображення. 2. Методи детекції порушення фокусування. 3. Методи детекції забруднення об'єктиву. 4. Методи детекції крапель води на об'єктиві. 5. Методи детекції затулення об'єктиву. 6. Методи детекції засвітлення об'єктиву. 7. Реалізація комплексного алгоритму оцінку якості зображення.

6. Перелік графічного (ілюстративного) матеріалу: Рисунки в тексті
пояснювальної записки, презентація до захисту

7. Консультанти розділів дисертації

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

8. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів роботи	Примітка
1	Літературний огляд матеріалу стосовно методів та засобів цифрового аналізу біомедичних зображень	10.03.2024	
2	Методи аналізу зображення	12.03.2024	
3	Опанування методів детекції порушення фокусування	22.03.2024	
4	Опанування методів детекції забруднення об'єктиву	02.04.2024	
5	Опанування методів детекції крапель води на об'єктиві	12.04.2024	
6	Опанування методів детекції затулення об'єктиву	24.04.2024	

7	Опанування методів детекції засвітлення об'єктиву	05.05.2024	
8	Реалізація комплексного алгоритму оцінку якості зображення	17.05.2024	
6	Проведення експериментальних досліджень запропонованих методів	28.05.2024	
7	Оформлення пояснювальної записки	03.06.2024	

Студент 
(підпис)

Васільєв М.О.
(прізвище та ініціали)

Науковий керівник работ 
(підпис)

Вунтесмері Ю.В.
(прізвище та ініціали)

РЕФЕРАТ

Дисертація містить основну частину на 82 аркушах, 22 ілюстрацій.

Метою наукового дослідження є розробка методів детекції порушення якості зображення для необслуговуваних або вбудованих камер спостереження з метою сигналізації про проблеми або керування моторизованими засобами підтримки.

Об'єктом дослідження є процес оцінки якості зображення з камер спостереження.

Предметом дослідження є алгоритми детекції порушень якості зображення, включаючи: методи детекції порушення фокусування, методи детекції забруднення об'єктиву, методи детекції крапель води на об'єктиві, методи детекції затулення та засвітлення об'єктиву.

Результатом роботи є:

- розробка алгоритмів детекції порушення фокусування, забруднення об'єктиву, крапель води на об'єктиві, накриття та засвітлення об'єктиву,
- інтеграція всіх методів в комплексний алгоритм, який перевіряє зображення по кожному критерію, дає висновок щодо наявності порушення якості зображення.

Галузь застосування: системи відеоспостереження, інтелектуальні транспортні системи, промисловий моніторинг, наукові проекти.

Ключові слова: порушення фокусування, забруднення об'єктиву, краплі води на об'єктиві, накриття об'єктиву, засвітлення об'єктиву, відеоспостереження.

ABSTRACT

The thesis contains the main part on 76 pages, 22 illustrations.

The goal of scientific research is to develop methods for detecting image quality violations for unattended or built-in surveillance cameras for the purpose of signaling problems or controlling motorized support means.

The object of research is the process of image quality assessment from surveillance cameras.

The subject of research are algorithms for detecting image quality violations, including: methods of detection of focusing failure, lens contamination detection methods, methods of detecting water drops on the lens, methods of detection of cloaking and illumination of the lens.

The result of the work is:

- development of algorithms for detection of focusing failure, contamination of the lens, water drops on the lens, covering and illumination of the lens,
- integration of all methods into a single algorithm that checks the image according to each criterion and gives a conclusion about the presence of a violation of image quality.

Field of application: video surveillance systems, intelligent transport systems, industrial monitoring, scientific projects.

Keywords: focusing failure, lens contamination, water drops on the lens, lens cover, lens illumination, video surveillance.

ЗМІСТ

ВСТУП.....	10
1 ФОРМУЛЮВАННЯ ПРОБЛЕМИ, АНАЛІТИЧНИЙ ОГЛЯД ТА МЕТОДИ АНАЛІЗУ ВІДЕОЗОБРАЖЕННЯ.....	15
Вступ.....	15
1.1 Актуальність атестаційної роботи.....	16
1.2 Формулювання мети і постановка основних задач роботи.....	17
1.3 Основні техніки та алгоритми обробки зображень.....	18
1.4 Програмне забезпечення та інструментарій для аналізу відео.....	20
2 МЕТОД ДЕТЕКЦІЇ ПОРУШЕННЯ ФОКУСУВАННЯ.....	23
Вступ.....	23
2.1 Порівняння і вибір методу детекції порушення фокусування.....	23
2.2 Опис методу Laplacian Variance.....	27
2.3 Реалізація алгоритму.....	29
2.4 Висновки до розділу.....	31
3 МЕТОД ДЕТЕКЦІЇ ЗАБРУДНЕННЯ ОБ'ЄКТИВУ.....	32
Вступ.....	32
3.1 Порівняння і вибір методу детекції забруднення об'єктиву.....	32
3.2 Опис методу Local Binary Patterns.....	36

3.3 Реалізація методу Local Binary Patterns.....	38
3.4 Оптимізація текстурного методу LBP.....	41
3.5 Висновки до розділу.....	44
4 МЕТОД ДЕТЕКЦІЇ КРАПЕЛЬ ВОДИ НА ОБ'ЄКТИВІ.....	45
Вступ.....	45
4.1 Існуючі методи детекції.....	46
4.2 Реалізація методу Хафоваго перетворення для колів.....	52
4.3 Висновки до розділу.....	55
5 МЕТОД ДЕТЕКЦІЇ НАКРИТТЯ ОБ'ЄКТИВУ.....	57
Вступ.....	57
5.1 Методика алгоритму для детекції накриття об'єктиву.....	58
5.1.1 Конвертація зображення в градації сірого.....	59
5.1.2 Порогова обробка (Thresholding).....	61
5.1.3 Пошук контурів (Finding Contours).....	61
5.1.4 Аналіз відсотка затулення.....	64
5.1.5	
Візуалізація.....	66
5.2 Реалізація та тестування методу.....	67
5.3 Висновки до розділу.....	71

6 МЕТОД ДЕТЕКЦІЇ ЗАСВІТЛЕННЯ ОБ'ЄКТИВУ	72
Вступ.....	72
6.1 Методика та реалізація алгоритму для детекції засвітлення об'єктиву.....	73
6.2 Тестування методу детекції засвітлення об'єктиву.....	76
6.3 Висновки до розділу.....	79
7 РЕАЛІЗАЦІЯ АЛГОРИТМУ ОЦІНКИ ЯКОСТІ ЗОБРАЖЕННЯ.....	81
ВИСНОВКИ.....	89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	92
ДОДАТКИ.....	95

ВСТУП

Якість зображення відіграє ключову роль у функціонуванні систем відеоспостереження, адже вона впливає на можливість ідентифікації осіб, розпізнавання предметів та загальну надійність системи безпеки. Давайте розглянемо основні аспекти цієї проблематики.

1. Вплив якості зображення на ефективність системи спостереження

Якість зображення безпосередньо впливає на здатність системи правильно ідентифікувати та реагувати на події в зоні спостереження. Недостатня різкість, контрастність або колірна вірність можуть істотно погіршити результати розпізнавання та аналізу ситуацій.

2. Загальні фактори, що впливають на якість зображення

- Роздільна здатність камери: Нижча роздільна здатність може призвести до того, що деталі будуть нерозрізненні, особливо на великих дистанціях.
- Компресія відео: Сильна компресія знижує якість зображення через втрату деталей і введення артефактів, що можуть перешкоджати ідентифікації.
- Умови освітлення: Недостатнє або надмірне освітлення може призвести до втрати деталей через затемнення або пересвітлення зображення.

3. Специфічні проблеми якості зображення

- Нерізкість через проблеми фокусування: Автоматичне фокусування може бути порушене забрудненням об'єктиву або його пошкодженням.

- Забруднення об'єктиву:Пил, бруд або краплі води на об'єктиві можуть зменшити чіткість зображення, призводячи до помилок у роботі системи.

- Механічні пошкодження об'єктиву: Подряпини або тріщини на об'єктиві змінюють шлях світла, що падає на сенсор, і викривлюють зображення.

- Ефекти засвітлення об'єктиву:Сильне зовнішнє світло, наприклад від лазера, може тимчасово або постійно пошкодити сенсор камери.

Для забезпечення високої ефективності систем спостереження необхідно звертати увагу на підтримання якості зображення на належному рівні. Розробка методів детекції та корекції згаданих вище проблем є важливим кроком до покращення надійності відеоспостереження.

Розвиток методів детекції якісних порушень зображення відіграє критичну роль у підвищенні надійності та ефективності систем камер спостереження. Оскільки ці системи часто використовуються для забезпечення безпеки та спостереження в різноманітних умовах, важливо, щоб камери постійно передавали чітке і точне зображення. Ось кілька ключових аспектів, що підкреслюють значення розробки цих методів:

- Забезпечення безпеки

Камери спостереження є життєво важливими для систем безпеки в громадських, комерційних та приватних просторах. Надійність зображень, які вони надають, може бути різницею між своєчасним виявленням злочинної діяльності та її пропуском. Методи детекції, які можуть виявити та виправити недоліки зображення, забезпечують, що системи спостереження залишаються ефективними навіть у складних умовах.

- Підвищення реакційної здатності системи

Камери, які автоматично виявляють проблеми з якістю зображення та адаптуються до змін умов, мають кращу здатність реагувати на зміни навколишнього середовища. Це може включати все від коригування налаштувань освітлення до активації додаткових систем очищення об'єктиву.

- Зменшення помилкових тривог

Системи, які можуть точно розрізнити проблеми якості зображення від дійсних загроз, зменшують кількість помилкових тривог. Наприклад, алгоритми, що відрізняють краплі води на об'єктиві від інтрузії у зону спостереження, підвищують точність системи та знижують непотрібні втручання персоналу.

- Економічна ефективність

Здатність камери самостійно виявляти та реагувати на проблеми з якістю зображення може значно знизити витрати на обслуговування та ремонт. Автоматизація цього процесу зменшує потребу в постійному моніторингу людиною, дозволяючи персоналу зосередитися на більш важливих завданнях.

- Інноваційний розвиток

Розвиток методів детекції порушення якості зображення стимулює технологічний прогрес в галузі обробки зображень і машинного навчання. Це веде до комплексного застосування, які можуть бути застосовані в інших технологічних сферах, зокрема в автомобільній промисловості та робототехніці.

Розробка та імплементація цих методів є критично важливою для підтримки високого рівня продуктивності та надійності сучасних систем відеоспостереження.

Ступінь вивченості теми

На сьогодні тема детекції порушення якості зображення в камерах спостереження активно розробляється, але все ще має ряд невирішених питань, що відкривають простір для подальших досліджень. Основні напрямки включають розробку алгоритмів для автоматичного виявлення різних видів дефектів зображення, таких як розфокусування, забруднення об'єктиву, або засвітлення. Попри значний прогрес у цій галузі, багато методів потребують покращення в контексті точності детекції та швидкості обробки даних. Важливою є також робота над інтеграцією цих технологій у різноманітні типи камер спостереження.

Ціль розробки

Головною метою даної розробки є створення комплексу алгоритмів для детекції порушень якості зображення, які можуть бути інтегровані в необслуговувані або вбудовані камери спостереження. Ці алгоритми повинні забезпечувати високу точність виявлення з урахуванням різноманітності

можливих порушень. Важливим є також зниження часу реакції систем на виявлені дефекти зображення, що дозволить оперативно корегувати параметри камери або попереджати операторів системи про потенційні проблеми.

Актуальність роботи

Актуальність розробки алгоритмів для детекції порушення якості зображення в камерах спостереження визначається кількома факторами:

- Підвищення безпеки: Висока точність і надійність камер спостереження є критично важливими для забезпечення безпеки в громадських та приватних зонах.
- Технологічний прогрес: Розвиток технологій обробки зображень відкриває нові можливості для покращення якості спостереження.
- Зниження витрат: Автоматизація процесу детекції дефектів зменшує потребу у постійному моніторингу та обслуговуванні систем, знижуючи оперативні витрати.
- Інноваційність: Розробка комплексного алгоритму детекції забезпечує технологічне лідерство в області систем безпеки, підвищуючи конкурентоздатність на ринку.

Ці аспекти підкреслюють необхідність продовження досліджень у даній області та впровадження нових рішень, що сприятиме зростанню якості та ефективності систем спостереження на всіх рівнях.

1. ФОРМУЛЮВАННЯ ПРОБЛЕМИ, АНАЛІТИЧНИЙ ОГЛЯД ТА МЕТОДИ АНАЛІЗУ ВІДЕОЗОБРАЖЕННЯ

Вступ

У цьому розділі ми розглянемо проблему детекції порушень якості зображень в системах відеоспостереження, яка є значною з огляду на зростаюче застосування автоматизованих систем спостереження в сучасних умовах. Розпізнавання та аналіз якості зображень відіграють вирішальну роль в ефективності систем спостереження, оскільки вони впливають на здатність системи ідентифікувати та реагувати на події в реальному часі [1].

Основна проблема полягає у виявленні та класифікації типів порушень, які можуть включати все, від нечіткості зображення і дефокусування до фізичних перешкод на об'єктиві, таких як вода, бруд або інші сторонні матеріали. Важливість цього дослідження підкреслюється роботами, такими як Zeng et al. (2012) [2], які розробили методи для автоматичного розпізнавання погіршення якості кадру в системах спостереження.

Аналітичний огляд літератури показує, що хоча існує значний прогрес у технологіях обробки зображень, проблема надійної детекції різних типів порушень якості залишається актуальною. Роботи, такі як та, що була представлена Wang et al. (2011) [3], вказують на складність вирішення проблеми через високу варіативність умов спостереження та потенційні

помилки, які можуть виникати під час автоматичної детекції.

В контексті методології це дослідження застосовує кілька підходів для оцінки якості зображення, включаючи як суб'єктивні, так і об'єктивні метрики. Важливість розробки надійних методів детекції підкреслюється в роботі Liu et al. (2021) [4], де використовується підхід на основі штучного інтелекту для покращення розпізнавання об'єктів в умовах поганої видимості.

Розробка ефективних методів детекції порушень якості зображення в камерах спостереження є критичною для підвищення загальної ефективності систем безпеки. Тому це дослідження спрямоване на вирішення цієї проблеми шляхом інтеграції передових технік обробки зображень та машинного навчання.

1.1 Актуальність атестаційної роботи

Сучасний розвиток міських інфраструктур та збільшення потреб у забезпеченні безпеки спонукають до впровадження більш складних та ефективних систем відеоспостереження. З якістю зображення, що забезпечується камерами спостереження, тісно пов'язані питання оперативної реакції на аварійні ситуації та правопорядок, а також ефективність роботи систем штучного інтелекту, що здійснюють аналіз відео в реальному часі. Наявність на об'єктиві вологи, бруду, крапель води або інших сторонніх предметів може суттєво погіршити якість зображення, що призводить до

зниження точності розпізнавання об'єктів та подій, а в деяких випадках — до повного невикористання даних відео.

Проблема якості зображення в камерах спостереження залишається актуальною, особливо в контексті розвитку смарт-систем та інтегрованих міських безпекових технологій. Розробка методів автоматизованої детекції та виправлення зазначених порушень якості має стратегічне значення для підвищення надійності та ефективності систем відеоспостереження в міських умовах.

1.2 Формулювання мети і постановка основних задач роботи

Мета атестаційної роботи — розробка комплексного алгоритму детекції порушень якості зображення для необслуговуваних або вбудованих камер спостереження, які можуть автоматично сигналізувати про проблеми або керувати моторизованими засобами підтримки.

Для досягнення цієї мети поставлено наступні основні задачі:

1. Аналіз існуючих методів детекції дефектів зображення в системах відеоспостереження з літературного огляду та дослідження існуючих технологій.
2. Розробка методології для виявлення специфічних порушень якості зображення, таких як дефокусування, забруднення об'єктиву, наявність крапель води, сторонніх частинок на об'єктиві, накриття (затулення)

об'єктиву та засвітлення об'єктиву (наприклад, лазером).

3. Розробка алгоритмічного забезпечення для реалізації детекції, що включає створення програмного забезпечення для обробки даних з камер спостереження.

4. Тестування та валідація розроблених методів на реальних та симульованих даних для оцінки ефективності та надійності розроблених рішень.

5. Аналіз впливу розроблених методів на загальну ефективність системи відеоспостереження та їхнє потенційне впровадження у реальні системи безпеки.

1.3 Основні техніки та алгоритми обробки зображень.

Зараз ми розглянемо ключові техніки та алгоритми, що застосовуються в обробці зображень, які є фундаментальними для розуміння сучасних систем відеоспостереження. Передобробка зображень (Pre-processing): - є критичним етапом у ланцюзі обробки зображень, яка включає такі методи як шумоподавлення, покращення контрасту, і корекція освітленості. Одним із популярних методів для шумоподавлення є медіанний фільтр, який допомагає видалити "сіль і перець" шум без значної втрати гостроти зображення. Формула медіанного фільтру виглядає так:

$$g(x, y) = \text{median}\{ f(x + i, y + j) : i, j \in W \} \quad (1)$$

, де $f(x, y)$ - вхідне зображення, $g(x, y)$ - вихідне зображення, а W - вікно фільтрування.

Сегментація зображень (Image Segmentation): Сегментація ділить зображення на сегменти або регіони, які є однорідними щодо деяких характеристик. Одним з найпопулярніших алгоритмів сегментації є алгоритм k-середніх, який використовується для кластеризації пікселів на основі їхньої інтенсивності та кольору. Формула алгоритму k-середніх:

$$C_k = \frac{1}{|S_k|} \sum_{x_i \in S_k} x_i \quad (2)$$

де C_k - центроїд кластера, S_k - множина пікселів у кластері k .

Виявлення країв (Edge Detection): детектори країв, такі як оператор Собеля, використовуються для виявлення місць з високою ступенем змін інтенсивності, що часто відповідають краям об'єктів. Собель використовує два набори ядер G_x і G_y для визначення градієнтів зображення по горизонталі і вертикалі. Формули оператора Собеля:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \quad (3)$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \quad (4)$$

, де A - вхідне зображення.

Морфологічні операції (Morphological Operations) - морфологічні операції, такі як дилатція (розширення) та ерозія, використовуються для обробки бінарних зображень. Ці операції допомагають виправляти дефекти у формі об'єктів та поліпшувати структуру зображення. Формули дилатції та ерозії:

$$(A \oplus B)(x, y) = \max_{(i,j) \in B} A(x - i, y - j) \quad (5)$$

$$(A \ominus B)(x, y) = \min_{(i,j) \in B} A(x - i, y - j) \quad (6)$$

, де A - вхідне зображення, B - структуруючий елемент.

Ці методи формують основу для розробки більш складних систем обробки відео, що використовуються в системах відеоспостереження для детекції і класифікації різноманітних подій і ситуацій. Використання цих технік є основою для підвищення точності та надійності систем спостереження, що є надзвичайно важливим в умовах постійного розвитку міських інфраструктур і потреби в забезпеченні високого рівня безпеки.

1.4 Програмне забезпечення та інструментарій для аналізу відео

У сучасному світі аналізу відеоданих програмне забезпечення на базі Python займає важливе місце завдяки своїй гнучкості та великій кількості бібліотек та інструментів. Python відомий своїми розширеними можливостями в обробці

даних, що робить його ідеальним для завдань аналізу відео.

OpenCV (Open Source Computer Vision Library): є однією з найпопулярніших бібліотек для обробки зображень і відео. Вона включає різноманітні інструменти для реалізації алгоритмів комп'ютерного зору, включаючи реалізацію передових алгоритмів машинного навчання, детекцію об'єктів, слідування за об'єктами та багато інших завдань [5]

SciPy та NumPy: ці бібліотеки є фундаментом для наукових обчислень в Python. Вони надають потужні інструменти для обробки та аналізу великих масивів даних, що є необхідним для обробки відео.

PyTorch і TensorFlow: ці фреймворки глибокого навчання використовуються для реалізації складних алгоритмів, таких як нейронні мережі для аналізу відео. Вони дозволяють автоматизувати багато завдань аналізу відео, включаючи класифікацію, розпізнавання та прогнозування.

MoviePy - це інструмент для редагування відео, спеціально розроблений для легкої роботи з відео в Python. MoviePy дозволяє вирізати частини, додавати титри, ефекти, анімації, звук і багато іншого, що робить його зручним інструментом для базового відео редагування та обробки.

Distant Viewing Toolkit (DVT) - цей пакет спеціально розроблений для аналізу візуальної культури. Він надає інструменти для квантитативного аналізу відео та зображень, зокрема детекцію об'єктів і подій в відео потоках [6]. DVT дозволяє виконувати складні аналізи візуального контенту, такі як визначення особливостей обличчя, розпізнавання об'єктів та динамічний аналіз сцен.

Ці інструменти Python забезпечують потужну підтримку для дослідників та розробників у поліпшенні та автоматизації аналізу відеоданих. Вони відкривають широкі можливості для створення інноваційних рішень у галузі відеоаналітики, від базової обробки зображень до реалізації складних алгоритмів машинного навчання та комп'ютерного зору.

Вивчення сучасних методів аналізу відеозображення підкреслює значний потенціал використання Python у даній роботі. Основна перевага Python полягає у його великій екосистемі потужних бібліотек для аналізу та обробки відеоданих, таких як OpenCV, SciPy, NumPy, PyTorch, TensorFlow, та інші. Ці інструменти надають необхідні засоби для швидкої та ефективної обробки відео, розробки алгоритмів машинного навчання, глибокого навчання, та комп'ютерного зору.

OpenCV виступає як одна з ключових бібліотек у даній області, пропонуючи розширений набір функцій для реалізації різних завдань комп'ютерного зору. PyTorch та TensorFlow дозволяють впроваджувати складні моделі глибокого навчання, які можуть ефективно працювати з великими масивами відеоданих. Водночас, SciPy та NumPy є незамінними для обробки та аналізу даних на науковому рівні.

Завдяки своїй універсальності, здатності інтегруватися з іншими мовами програмування та підтримці великої спільноти, Python є оптимальним вибором для реалізації проектів в галузі аналізу відеозображень у цій магістерській роботі. Це забезпечує не тільки технічну ефективність, але й значне зниження часу розробки та покращення якості кінцевих результатів аналізу.

2 МЕТОД ДЕТЕКЦІЇ ПОРУШЕННЯ ФОКУСУВАННЯ

Вступ

Один з підходів до виявлення порушень фокусування у системах відеоспостереження полягає у використанні методу оцінки різкості зображення за допомогою аналізу його частотних компонентів. Цей метод базується на припущенні, що нерізкі зображення мають менш виражені високочастотні компоненти порівняно з різкими зображеннями.

2.1 Порівняння і вибір методу детекції порушення фокусування

Детекція порушення фокусування є важливою задачею в різних областях, включаючи обробку зображень, комп'ютерний зір, медичну діагностику та контроль якості виробництва. Ось деякі з основних методів, які використовуються для детекції порушення фокусування:

Метод Laplacian Variance - використовує оператор Лапласа для аналізу змін яскравості пікселів на зображенні. Оператор Лапласа підкреслює місця, де яскравість змінюється найбільш різко, як-от краї об'єктів. Коли зображення добре сфокусоване, такі зміни будуть значними і частими, що призводить до високої варіації значень Лапласіана. Метод обчислює середнє значення і

дисперсію цих змін. Висока дисперсія свідчить про добре сфокусоване зображення, оскільки показує велику кількість різких переходів яскравості.

Процедура:

1. Виділення країв: Використовується оператор Лапласа для виявлення різких переходів у яскравості.
2. Обчислення дисперсії: Визначається, наскільки сильно змінюються ці переходи по всьому зображенню.
3. Аналіз результатів: Високі значення дисперсії вказують на добре сфокусоване зображення.

Метод Brenner - оцінює фокусування зображення шляхом порівняння яскравості пікселів на певній відстані один від одного. Він вимірює різницю яскравості між пікселями, які знаходяться на два кроки один від одного по горизонталі. Коли зображення добре сфокусоване, ці різниці будуть значними, оскільки краї об'єктів будуть чіткими і яскравість змінюватиметься різко. Метод підсумовує всі ці різниці по всьому зображенню.

Процедура:

1. Вимірювання різниці яскравості: Для кожної пари пікселів, що знаходяться на певній відстані один від одного, обчислюється різниця їх яскравості.
2. Підсумування різниць: Усі отримані різниці підсумовуються для всього зображення.
3. Аналіз результатів: Високі значення сумарної різниці вказують на добре

сфокусоване зображення.

Метод Tenengrad - базується на аналізі градієнтів зображення. Градієнт показує, наскільки швидко змінюється яскравість пікселів у певному напрямку (горизонтально або вертикально). Цей метод використовує фільтри Собела для обчислення градієнтів для кожного пікселя зображення. Величина градієнта вказує на різкість змін яскравості. Коли зображення добре сфокусоване, величини градієнтів будуть великими, оскільки різкі краї об'єктів створюють значні зміни в яскравості. Метод підсумовує всі величини градієнтів по всьому зображенню.

Процедура:

1. Обчислення градієнтів: Використовуються фільтри Собела для визначення зміни яскравості у горизонтальному та вертикальному напрямках.
2. Вимірювання величин градієнтів: Обчислюється величина градієнта для кожного пікселя, що вказує на різкість зміни яскравості.
3. Підсумування величин: Всі величини градієнтів підсумовуються для всього зображення.
4. Аналіз результатів: Високі значення сумарної величини градієнтів свідчать про добре сфокусоване зображення.

Ці три методи забезпечують різні підходи до оцінки фокусування зображень, аналізуючи зміни яскравості або градієнтів для визначення різкості та чіткості зображення. У цьому розділі ми розглянемо саме метод Laplacian Variance для детекції порушення фокусування, оскільки він є найкращим серед інших

методів, таких як метод Brenner і метод Tenengrad. Метод Laplacian Variance має декілька ключових фізичних переваг, які роблять його найбільш ефективним та надійним для оцінки різкості зображень:

1. Висока чутливість до змін яскравості:

- Оператор Лапласа чутливий до найменших змін яскравості пікселів, що дозволяє точно визначати різкі переходи на краях об'єктів. Це робить метод особливо ефективним для виявлення добре сфокусованих зображень, де такі переходи найбільш виражені.

2. Аналіз глобальної різкості:

- Метод Laplacian Variance враховує зміну яскравості по всьому зображенню, що дозволяє отримати глобальну оцінку різкості. Це є важливою перевагою порівняно з методами, що базуються на локальних змінах, оскільки забезпечує більш точну та надійну оцінку якості зображення в цілому.

3. Швидкість і ефективність обчислень:

- Обчислення Лапласіана та його дисперсії є відносно простими та швидкими, що дозволяє використовувати метод Laplacian Variance в реальному часі. Це особливо важливо для застосувань, де потрібна швидка та надійна оцінка різкості, наприклад, у системах автоматичного фокусування або контролю якості.

4. Відсутність впливу шуму:

- Метод Laplacian Variance менш чутливий до шуму у порівнянні з іншими методами, такими як методи, що базуються на градієнтах. Це зменшує

ймовірність хибних спрацьовувань та забезпечує більш стабільну роботу в умовах низької якості зображення або в присутності артефактів.

Зважаючи на наведені фізичні переваги, метод Laplacian Variance є оптимальним вибором для детекції порушення фокусування. У цьому розділі ми детально розглянемо теоретичні аспекти цього методу, а також його практичне застосування для оцінки різкості зображень у різних галузях.

2.2 Опис методу Laplacian Variance

Метод "Laplacian Variance" (Варіативність Лапласіана) є одним з ефективних підходів для автоматичного визначення різкості зображень. Цей метод використовується для аналізу різкості зображень у великій кількості застосувань, включаючи системи відеоспостереження та медичну візуалізацію. Варіативність Лапласіана особливо корисна для визначення того, чи є зображення різким або замиленим [8].

Лапласіан є другою похідною зображення, і він широко використовується для виявлення країв. Це тому, що краї на зображенні, як правило, кореспондують з різкими перепадами інтенсивності, які легко виявляються другими похідними. Лапласіан можна описати наступною формулою:

$$\nabla^2 I = \frac{\partial^2 I}{\partial^2 x^2} + \frac{\partial^2 I}{\partial^2 y^2} \quad (7)$$

, де I є зображенням, а x та y є просторовими координатами.

Обчислення варіативності Лапласіана - варіативність Лапласіана обчислюється як дисперсія (варіативність) значень Лапласіану по всьому зображенню. Вона вимірює ступінь різкості або контрастності країв, присутніх у зображенні. Математично, вона визначається так:

$$LV = \sigma^2(\nabla^2 I) \quad (8)$$

, де $\nabla^2 I$ - лапласіан зображення I , а σ^2 - дисперсія значень лапласіану.

Застосування методу

1. Передобробка: Зображення конвертується до сірого масштабу, якщо це необхідно, для спрощення аналізу.
2. Фільтрація: На зображення застосовується лапласіанський фільтр для отримання карти країв.
3. Обчислення дисперсії: Обчислюється дисперсія значень отриманої карти країв.

Практичне використання - метод варіативності Лапласіана використовується для автоматичної діагностики різкості в системах відеонагляду, де важливо швидко ідентифікувати проблеми з фокусуванням. Наприклад, у системах безпеки можна автоматично повідомляти операторів про потенційні проблеми з різкістю зображення, що дозволяє своєчасно коригувати налаштування обладнання або вживати інших заходів.

Цей метод є ефективним інструментом для забезпечення якості зображення в різних прикладних сферах, де важлива різкість зображення, забезпечуючи

надійність та точність обробки візуальної інформації.

2.3 Реалізація алгоритму

Для реалізації методу варіативності Лапласіана, який дозволяє визначити рівень фокусування зображення, можна використовувати бібліотеку OpenCV у Python. Цей метод оцінює чіткість зображення шляхом аналізу варіації значень другої похідної (Лапласіан) якості зображення.

Підхід до реалізації:

1. Завантажити зображення.
2. Перетворити зображення у відтінки сірого, оскільки аналіз чіткості не потребує кольорової інформації.
3. Застосувати Лапласіан з використанням функції `'cv2.Laplacian'` для отримання другої похідної кожного пікселя.
4. Обчислити варіативність Лапласіана (тобто дисперсію значень Лапласіана) для оцінки чіткості.
5. Виходом буде числове значення, яке представляє чіткість зображення: чим вище значення, тим більша чіткість.

Реалізуємо та напишемо код на мові Python для визначення чіткості зображення за допомогою методу варіативності Лапласіана (Рисунок 2.1). Ось основні кроки, які були реалізовані:

```

2.1 focus.py x
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 def display_image_with_variance(image_path):
6     # Завантажити зображення
7     image = cv2.imread(image_path)
8     # Конвертувати зображення у відтінки сірого
9     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10    # Обчислити Лапласіан зображення
11    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
12    # Варіативність Лапласіана (дисперсія значень Лапласіана)
13    variance = laplacian.var()

```

Рисунок 2.1 – Реалізація методу варіативності

Ця функція може приймати шлях до зображення як аргумент і повертати числове значення, яке вказує на чіткість зображення. Вище значення варіативності свідчить про більшу чіткість. Лістинг коду програми буде показано в Додатку А. Результат надано на Рисунку 2.2 :

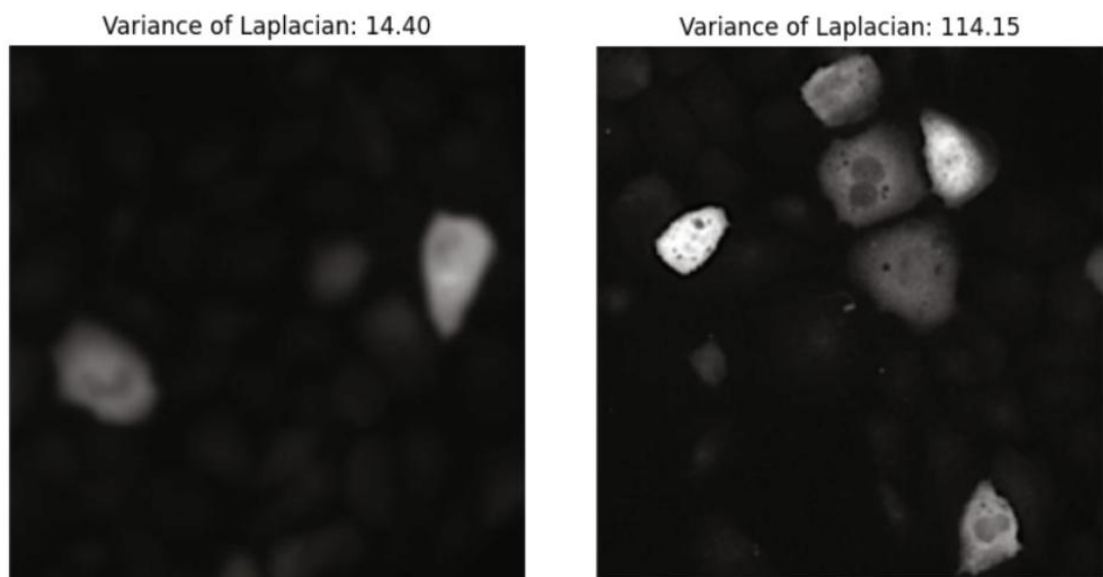


Рисунок 2.2 – Результат обчислення варіативності

2.4 Висновки

Як видно з Рисунку 2.2, наші результати відповідають теорії яка була розглянута, адже більшому фокусуванню відповідає більше значення варіативності. Стале значення варіативності Лапласіана, за якого зображення вважається чітким, зазвичай визначається на основі емпіричних досліджень та може варіюватися в залежності від специфіки камери, умов зйомки та параметрів обробки зображення. Це не стандартизований показник, і він може бути різним для різних систем відеоспостереження.

Для точного визначення порогового значення, за якого зображення вважається чітким, необхідно провести серію тестів на репрезентативному наборі зображень. Тестування має включати аналіз різних рівнів розмитості та визначення відповідності візуальної різкості з обчисленими значеннями LV .

3 МЕТОД ДЕТЕКЦІЇ ЗАБРУДНЕННЯ ОБ'ЄКТИВУ

Вступ

В умовах зростаючого застосування систем відеоспостереження в промисловості, на транспорті та в громадських просторах, підтримка високої якості зображення стає все більш актуальною задачею. Забруднення об'єктиву камери може серйозно вплинути на ефективність відеоспостереження, роблячи зображення непридатними для аналізу та прийняття рішень. У зв'язку з цим, розробка та впровадження методів автоматичного виявлення забруднення об'єктиву є критично важливою.

На сьогоднішній день існує багато методів, що дозволяють аналізувати забрудненість об'єктиву, варіюючи від простих алгоритмів аналізу змін у яскравості та контрастності зображення до складних методів машинного навчання, які використовують глибокі нейронні мережі. Вибір конкретного методу залежить від специфіки задачі, доступних ресурсів та вимог до точності та швидкості обробки.

3.1 Порівняння і вибір методу детекції забруднення об'єктиву

Метод Local Binary Patterns (LBP) - використовується для аналізу текстур зображень та виявлення структурних особливостей. Цей метод розглядає кожен

піксель зображення та його сусідів для визначення локального шаблону, який описує текстуру в цій області.

Процедура:

1. Локальний аналіз: Для кожного пікселя обирається невелика область сусідніх пікселів (наприклад, 3×3).
2. Порівняння з центром: Яскравість кожного сусіднього пікселя порівнюється з яскравістю центрального пікселя. Якщо сусідній піксель яскравіший або рівний центральному, йому присвоюється значення 1, інакше 0.
3. Формування шаблону: Бінарні результати порівнянь формуються в двійковий код (шаблон), який представляє текстурні характеристики цієї області.
4. Гістограма шаблонів: Для всього зображення створюється гістограма отриманих двійкових шаблонів, яка описує текстурні особливості зображення.

Метод аналізу гістограми - аналізує розподіл яскравості пікселів на зображенні. Гістограма представляє собою графік, де кожне значення яскравості відповідає кількості пікселів з цією яскравістю.

Процедура:

1. Побудова гістограми: Яскравість кожного пікселя зображення підраховується та заноситься в гістограму.

2. Аналіз розподілу: Виявляються аномалії або нерівномірності в розподілі яскравостей. Наприклад, забруднення об'єктиву може призводити до появи несподіваних піків або провалів на гістограмі.

3. Інтерпретація результатів: Аналізуючи форму гістограми, можна зробити висновки про наявність забруднення або інших артефактів на об'єктиві.

Метод спектрального аналізу - використовує перетворення Фур'є або інші спектральні методи для аналізу частотного вмісту зображення. Він дозволяє визначити, які частоти (деталі) присутні на зображенні та з якою інтенсивністю.

Процедура:

1. Перетворення Фур'є: Зображення перетворюється з просторової області в частотну область. Це дозволяє аналізувати зображення на рівні частот.

2. Аналіз частотного спектру: Визначаються інтенсивності різних частотних компонентів. Забруднення об'єктиву часто призводить до появи низькочастотних артефактів або зменшення високочастотних компонентів.

3. Інтерпретація результатів: Аналізуючи спектральний вміст, можна виявити наявність забруднення об'єктиву, яке проявляється у вигляді специфічних змін у частотному спектрі.

Ці методи забезпечують різні підходи до виявлення забруднення об'єктиву, використовуючи текстурний аналіз, розподіл яскравостей або частотний вміст зображення. У цьому розділі ми розглянемо метод Local Binary Patterns (LBP) для детекції забруднення об'єктиву, оскільки він має декілька ключових переваг, які роблять його найбільш ефективним та надійним для виявлення забруднень на об'єктиві.

- Стійкість до змін освітлення, LBP менш чутливий до змін у освітленні, оскільки базується на відносних значеннях яскравості пікселів у локальних областях. Це дозволяє надійно виявляти текстурні зміни, викликані забрудненнями, незалежно від умов освітлення.

- Локальний аналіз текстур, метод LBP фокусується на аналізі локальних текстур, що робить його ідеальним для виявлення дрібних дефектів та аномалій на поверхні об'єктиву. Забруднення, такі як пил або плями, створюють специфічні текстурні патерни, які легко виявити за допомогою LBP.

- Висока швидкість обробки, обчислення LBP є відносно простим і швидким процесом, що дозволяє використовувати цей метод у реальному часі. Це особливо важливо для застосувань, де необхідна миттєва реакція на виявлення забруднень, наприклад, у системах відеоспостереження або контролю якості.

- Ефективність у виявленні структурних особливостей, LBP ефективно виявляє структурні особливості зображення, такі як краї, кути та текстури. Це дозволяє точно визначати навіть незначні забруднення на об'єктиві, які можуть залишитися непоміченими іншими методами.

Зважаючи на наведені переваги, метод Local Binary Patterns (LBP) є оптимальним вибором для детекції забруднення об'єктиву. У цьому розділі ми детально розглянемо теоретичні аспекти цього методу, а також його практичне застосування для виявлення дефектів та аномалій на об'єктивах.

3.2 Опис методу Local Binary Patterns

Local Binary Patterns (LBP) — це потужний, надійний і відносно простий у реалізації метод текстурного аналізу, який використовується у багатьох застосуваннях комп'ютерного зору, включаючи аналіз забруднення об'єктів камер. LBP був розроблений Тімо Ояла і Матті Піетікайнен у 1990-х роках і з тих пір знайшов широке застосування у різних областях [9-10].

Основна концепція - LBP оцінює місцеві структури зображення, аналізуючи кожен піксель і його сусідів. Сусіди пікселя порівнюються з ним, і на основі цього порівняння формується двійковий код, який відображає місцеву текстурну структуру.

Принцип роботи:

1. Вибір центрального пікселя: LBP виконується шляхом вибору кожного пікселя зображення як центру.
2. Порівняння з сусідами: Сусідні пікселі (зазвичай вибирають 8 сусідів) порівнюються з центральним пікселем. Якщо інтенсивність сусіднього пікселя більше або рівна центральному, він отримує значення 1, інакше 0.
3. Формування двійкового числа: Ці бінарні значення об'єднуються в годинниковому або протигодинниковому напрямку для формування двійкового числа. Формула LBP:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c) \cdot 2^p \quad (9)$$

, де g_c є значенням центрального пікселя, g_p - значеннями сусідніх пікселів, P - кількістю сусідніх пікселів, R - радіусом навколо центрального пікселя, і $s(x)$ є функцією, яка визначається як:

- $s(x) = 1$, якщо $x \geq 0$ - це означає, що якщо інтенсивність сусіднього пікселя більша або дорівнює інтенсивності центрального пікселя, то результат буде 1.

- $s(x) = 0$, якщо $x \leq 0$ - це означає, що якщо інтенсивність сусіднього пікселя менша за інтенсивність центрального пікселя, то результат буде 0.

Ця функція є критично важливою для створення двійкового числа, яке відображає місцевий шаблон текстури навколо кожного пікселя на зображенні. Кожен піксель і його восьми сусідів аналізуються з використанням цієї функції, і результати об'єднуються в одне двійкове число, що служить як унікальний ідентифікатор текстурного шаблону в тому місці зображення.

Застосування до детекції забруднення об'єктиву

У контексті детекції забруднення об'єктиву, LBP може виявляти неоднорідності на поверхні об'єктиву, які можуть відображатися як зміни в локальних текстурних шаблонах. Наприклад, відкладення пилу або крапель води можуть змінити місцеву текстурну характеристику, що виявляється через зміни у LBP характеристиках.

3.3 Реалізація методу Local Binary Patterns

Для реалізації детекції забруднення об'єктиву на основі аналізу текстур можемо використати метод Local Binary Patterns. LBP ефективний для аналізу текстурних властивостей зображень і часто використовується в задачах виявлення змін у зображеннях. Він дозволяє перетворити кожен піксель зображення у бінарний код, який відображає локальну текстуру навколо нього.

Етапи реалізації:

1. Передобробка зображення - перетворення зображення у відтінки сірого.
2. Застосування LBP - генерація LBP-карти, яка відображає текстурні особливості.
3. Аналіз розподілу LBP-карти — використання статистичних методів для аналізу LBP-карти та виявлення аномалій у текстурних особливостях.
4. Локалізація зон забруднення — використання порогової обробки або кластеризації для визначення зон на зображенні, які мають аномальні текстурні характеристики.

Реалізуємо даний метод в середовищі PyCharm та виведемо результат на Рисунок 3.1, також код програми буде в Додатку Б

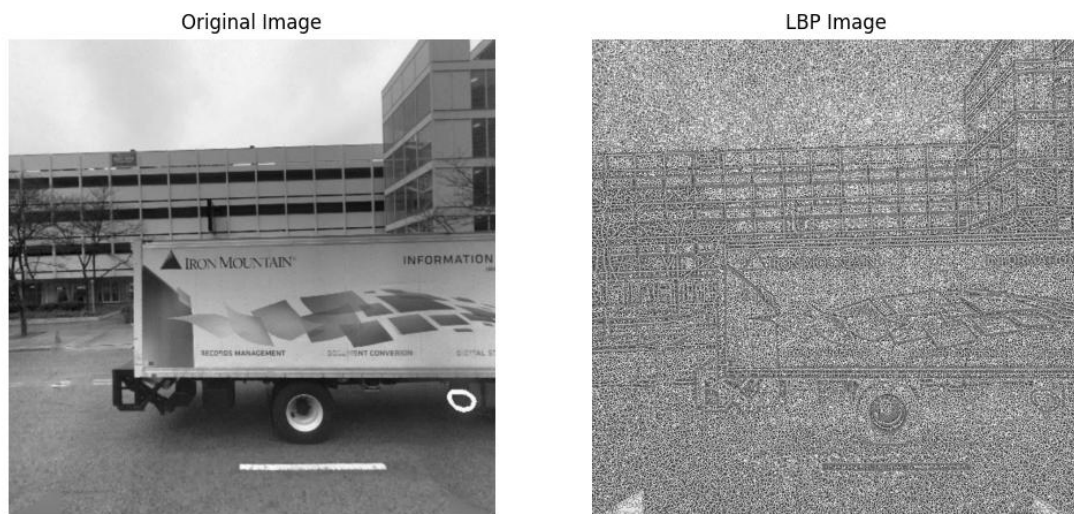


Рисунок 3.1 – Перетворення початкового зображення в LBP - карти

Для розширення існуючої функціональності, щоб точніше ідентифікувати та візуалізувати області забруднення на зображенні, нам потрібно додати наступні кроки:

1. Аналіз гістограми LBP: Визначення порогового значення для відділення забруднених ділянок на основі їх LBP-характеристик.
2. Бінаризація зображення: Перетворення LBP-зображення в бінарне зображення, де "1" відповідає забрудненим областям, а "0" — чистим.
3. Морфологічні операції: Застосування морфологічних операцій для покращення візуалізації забруднених областей.
4. Визначення наявності забруднення: Аналіз бінарного зображення для визначення, чи є на камері забруднення.
5. Візуалізація результатів: Підсвічування забруднених областей на

оригінальному зображенні.

Цей підхід дозволяє не тільки визначити наявність забруднення, але й локалізувати його на зображенні, що може бути корисно для систем автоматичного контролю якості зображення в різних застосуваннях.

Отже, реалізуємо це в нашому середовищі PyCharm (вихідний код програми - Додаток В), виведемо результати на Рисунку 3.2 та проаналізуємо отримані результати:

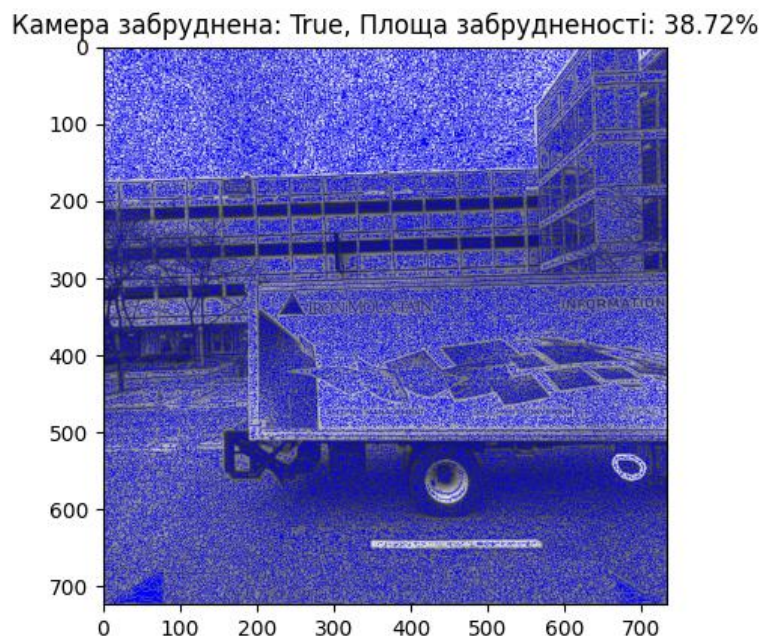


Рисунок 3.2 – Результат ідентифікації рівня забрудненості об'єктиву

Результат виконання коду показує, що забруднення присутнє на зображенні, і візуалізація забруднених ділянок вказує на значну частину зображення, оцінювану як забруднену (близько 38.72%). Варто зазначити, що метод виявляє забруднення на основі текстурних особливостей та може бути чутливим до різноманітних текстур у зображенні, що може призвести до хибних результатів.

Тому такий висновок може вимагати додаткової перевірки або налаштування порогових значень та інших параметрів алгоритму.

Також, цей підхід може давати помилкові спрацьовування на зображеннях з великою кількістю деталей і контрастних переходів, як у випадку з автомобілем на зображенні. Для більш точної роботи алгоритму можуть знадобитися додаткові налаштування та оптимізація.

3.4 Оптимізація текстурного методу LBP

Щоб зменшити кількість хибнопозитивних спрацьовувань і підвищити точність методу, ми можемо вдосконалити наш підхід. Ось кілька можливих способів для оптимізації:

1. Фільтрація Шумів: Застосування фільтрації для зменшення впливу текстурного шуму, наприклад, медіанного або гауссового фільтра.
2. Покращена Бінаризація: Використання адаптивної бінаризації, яка може краще враховувати локальні особливості зображення.
3. Морфологічне Відкриття та Закриття: Застосування морфологічних операцій для видалення дрібних об'єктів і заповнення невеликих дірок.
4. Площа забруднення: Встановлення порогу за площею забруднення, щоб відсікти дрібні забруднення, які можуть бути незначними.

Перейдемо до оптимізації , остаточний код буде наведено в Додатку Г , а

результат обробленого зображення буде представлено на Рисунку 3.3:

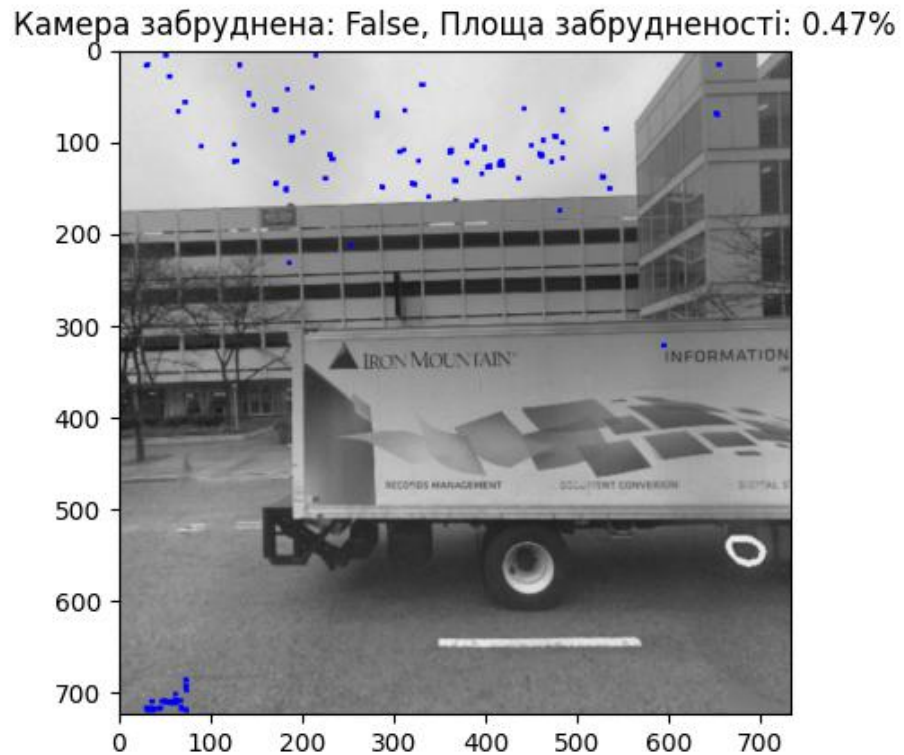


Рисунок 3.3 – Оптимізований текстурний метод детекції

Завдяки оптимізації алгоритму, ми тепер маємо результат, що виділяє значущі області забруднення. На останньому зображенні, алгоритм визначив 0.47% забруднення від загальної площі зображення. Як результат - виводиться вказівка чи забруднений об'єктів True або False.

Також є можливість налаштувати поріг спрацьовування індикації забруднення, в програмі виставляється критичне значення забрудненості.

Для наглядності продемонструємо роботу алгоритму на інших зображеннях, та представимо на наступних рисунках.

Камера забруднена: False, Площа забрудненості: 1.64%

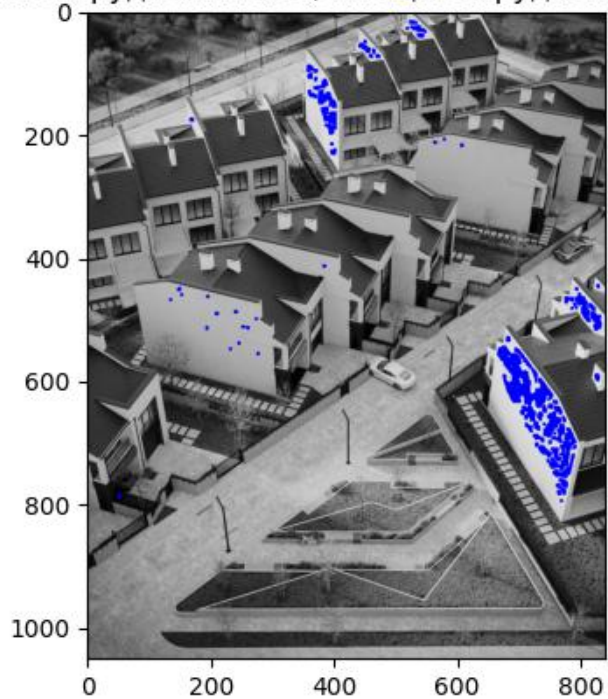


Рисунок 3.4 – Приклад деталізованого зображення на наявність забруднення

Камера забруднена: True, Площа забрудненості: 28.59%

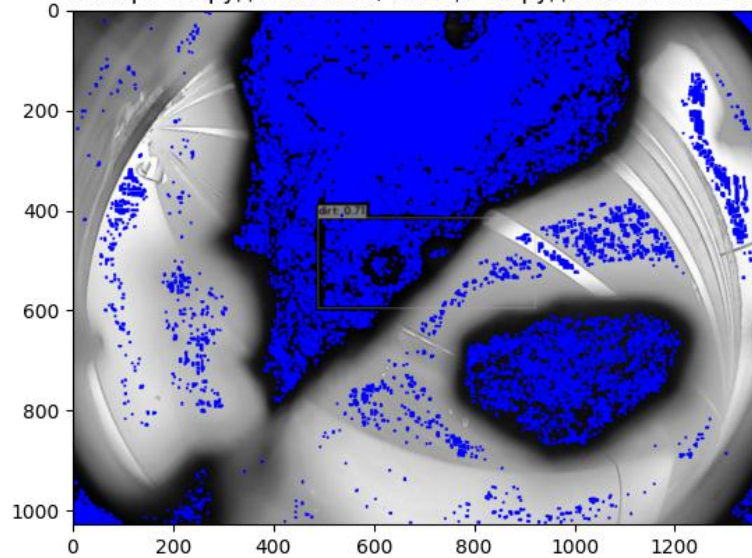


Рисунок 3.5 – Зображення із явним забрудненням об'єктиву

3.5 Висновки

Аналізуючи останні три зображення , а саме Рисунки 3.3 , 3.4 та 3.5 - видно наскільки краще відпрацьовує алгоритм детекції забруднення об'єктиву, після його вдосконалення ми отримуємо позитивні результати, які можна і корегувати відповідно до поставлених задач, а також під особливості камери.

4 МЕТОД ДЕТЕКЦІЇ КРАПЕЛЬ ВОДИ НА ОБ'ЄКТИВІ

Вступ

Введення крапель води на об'єктив камери може істотно вплинути на якість зображення, що отримується системами відеоспостереження. Це особливо критично в умовах, коли камери використовуються для безпеки, навігації або інших відповідальних застосувань, де якість зображення є необхідністю. Краплі води можуть спотворювати зображення, створюючи розмиті або спотворені ділянки, що ускладнює ідентифікацію осіб, автомобілів або інших важливих об'єктів.

Розробка ефективних методів для автоматичного виявлення крапель води на об'єктиві є суттєвою задачею, що допомагає підтримувати надійність і ефективність систем відеонагляду. Виявлення і адекватне реагування на такі проблеми можуть забезпечити своєчасне усунення забруднень з об'єктиву або активувати відповідні системи очищення, що знижує ризики пов'язані з втратою візуальної інформації. Сучасні підходи до детекції крапель води включають ряд методів, що базуються на аналізі змін у візуальних характеристиках зображення, використанні спеціалізованого алгоритмічного забезпечення та застосуванні технік машинного навчання. Ці методи спрямовані на те, щоб точно ідентифікувати наявність крапель води та оцінити їхній вплив на якість зображення.

Тому в цьому розділі розглянемо метод детекції крапель води та реалізованого його на мові Python.

4.1 Існуючі методи детекції

Хафове перетворення для колів є одним з найвідоміших алгоритмів у комп'ютерному зорі для виявлення круглих форм. Цей метод є особливо ефективним для виявлення крапель води на об'єктиві камери, завдяки їх природній круглій формі, яка створює характерні спотворення на зображенні [10].

Основний принцип - Хафове перетворення працює шляхом перетворення кожної точки зображення в простір параметрів, де кожен круг на зображенні представлений як точка. Це перетворення дозволяє ефективно агрегувати голоси за потенційні центри та радіуси кіл, що відповідають виявленим формам.

Алгоритм виконання:

1. Попередня обробка зображення: Зазвичай зображення перетворюється до сірого масштабу і використовується фільтр для зменшення шуму, такий як Гаусівський розмиття.
2. Виявлення країв: Застосування алгоритму виявлення країв, такого як оператор Канні, для ідентифікації країв на зображенні.

3. Хафове перетворення: Кожна точка з виявленими краями розглядається як можливий край круга. Для кожної такої точки в просторі параметрів акумулюються можливі центри і радіуси.

4. Визначення кіл: Найбільші акумулятори в просторі параметрів вказують на центри і радіуси найбільш ймовірних кіл, що відповідають краплям води.

Формула Хафового перетворення для кола - для кожної точки краю на зображенні (x, y) і потенційного радіуса r , точки в просторі параметрів (a, b) обчислюються як:

$$a = x - r \cdot \cos(\theta) \quad (10)$$

$$b = y - r \cdot \sin(\theta) \quad (11)$$

, де θ варіюється від 0 до 2π .

Цей метод є дуже ефективним для систем, яким необхідно виявляти та реагувати на краплі води на об'єктивах у реальному часі, забезпечуючи якість зображення та надійність моніторингу.

Використання порогової обробки та контурного аналізу для детекції крапель води на об'єктиві - основна ідея методу порогової обробки та контурного аналізу є класичним підходом в обробці зображень для сегментації об'єктів і виявлення їхніх форм. В контексті детекції крапель води на об'єктиві, цей метод дозволяє ефективно відокремити краплі води від фону зображення за допомогою порогової обробки, а потім використовувати аналіз контурів для ідентифікації характерних круглих форм крапель.

Методика

1. Порогова обробка:

- Перетворення зображення до сірого масштабу (якщо воно кольорове).
- Застосування порогової обробки для виділення областей з високою інтенсивністю, що відповідають краплям води. Це може бути зроблено за допомогою глобального порогу або адаптивних методів порогової обробки.

Порогова обробка зображення полягає в перетворенні зображення до бінарного вигляду на основі порогу T . Для кожного пікселя з інтенсивністю $I(x, y)$ виконується наступне:

$$I'(x, y) = \begin{cases} 1, & \text{якщо } I(x, y) \geq T \\ 0 & \text{в інших випадках} \end{cases} \quad (12)$$

, де $I'(x, y)$ - значення пікселя в бінарному зображенні, T - порогове значення.

2. Контурний аналіз:

- Використання алгоритмів знаходження контурів, таких як алгоритм Canny для виявлення країв який включає кілька етапів, але основне рівняння для градієнта G на основі ядер Собеля:

$$G = \sqrt{G_x^2 + G_y^2} \quad (13)$$

, де G_x та G_y описані у першому розділі в формулах (3) та (4) відповідно.

- Застосування алгоритмів пошуку контурів (наприклад, за допомогою функцій OpenCV `findContours`), для ідентифікації окремих областей.

- Аналіз форм контурів для виявлення тих, що мають круглу форму, характерну для крапель води.

Принципи роботи порогової обробки - перетворює оригінальне зображення у бінарне, де краплі води виділяються як білі області на темному фоні. Після цього, контурний аналіз дозволяє виділити та аналізувати форму цих областей, визначаючи, чи відповідають вони геометричним параметрам крапель води.

Сегментація на основі кольору та текстури для детекції крапель води на об'єктиві. Основна ідея методу - Сегментація на основі кольору та текстури є ефективним методом виявлення крапель води на об'єктиві, особливо коли краплі мають відмінності в кольорі або текстурі порівняно з фоном. Цей метод включає аналіз зображення для ідентифікації регіонів, що відповідають критеріям вибраного діапазону кольорів або специфічним текстурним особливостям.

Принципи роботи

1. Визначення характеристик - аналізується зображення для визначення характеристичних кольорових або текстурних властивостей крапель води. Наприклад, краплі води часто мають відблискуючі властивості або виділяються на темному фоні.
2. Сегментація кольору - використовуються кольорові простори, такі як HSV (Hue, Saturation, Value), для ефективнішої сегментації кольорових відмінностей. Це дозволяє ізолювати краплі води, які можуть мати відмінну насиченість або відтінок.

3. Текстурна сегментація - застосування алгоритмів для аналізу текстури, таких як Local Binary Patterns (LBP) або Gabor filters, дозволяє виділити краплі води на основі їхніх текстурних особливостей, які відрізняються від загального фону зображення.

Технічне виконання - основні математичні формули, які використовуються в методі сегментації на основі кольору та текстури для детекції крапель води на об'єктиві, включають перетворення кольорового простору, порогову обробку та морфологічні операції [11]. Ось детальний опис кожної з цих формул:

1. Перетворення кольорового простору

Перехід з RGB до HSV допомагає краще ідентифікувати кольорові відмінності, особливо коли краплі води мають відблиск або відмінний колір:

$$\text{Image}_{\text{HSV}} = \text{convertColor}(\text{Image}_{\text{RGB}}, \text{COLOR_RGB2HSV}) \quad (14)$$

Ця функція перетворює зображення з кольорової моделі RGB (Червоний, Зелений, Блакитний) в HSV (Відтінок, Насиченість, Значення), що робить краплі води більш розрізненими на основі їх кольорових характеристик.

2. Порогова обробка

Порогова обробка в HSV просторі використовується для виділення крапель води на основі відтінку та насиченості:

$$\text{Mask} = (\text{Hue}_{\min} \leq \text{Hue} \leq \text{Hue}_{\max}) \& (\text{Sat}_{\min} \leq \text{Sat} \leq \text{Sat}_{\max}) \quad (15)$$

де:

- Hue_{min}, Hue_{max} - мінімальне та максимальне значення відтінку для визначення крапель.

- Sat_{min}, Sat_{max} - мінімальне та максимальне значення насиченості.

Ці пороги дозволяють ізолювати області зображення, які відповідають кольоровим характеристикам крапель води.

3. Морфологічні операції

Морфологічні операції, такі як диляція та ерозія, використовуються для покращення маски, отриманої після порогової обробки:

Диляція

$$\text{Dilation}(I) = I \oplus B \quad (16)$$

, де B є структурним елементом, I - вхідне зображення, а $\oplus$ позначає диляцію, яка розширює об'єкти.

Ерозія

$$\text{Erosion}(I) = I \ominus B \quad (17)$$

, де $\ominus$ - позначає ерозію, яка зменшує розміри об'єктів.

Ці операції допомагають усунути малі шуми та закріпити більші об'єкти в масці, що покращує точність виявлення крапель води.

У наступному пункті цього ми розглянемо реалізацію Хафоваго

перетворення для колів (Hough Circle Transform). Цей метод є ефективним інструментом для автоматичного виявлення круглих об'єктів, таких як краплі води на зображенні і має такі основні переваги порівняно з іншими методами:

- Висока точність виявлення кіл, метод Hough Circle Transform здатен точно виявляти кола навіть у складних сценах з великою кількістю об'єктів та шуму. Він ефективно знаходить геометричні форми, що відповідають заданим параметрам.

- Можливість виявлення кіл різного розміру, метод дозволяє одночасно шукати кола різних розмірів, що особливо корисно для аналізу зображень, де можуть бути присутні кола з різними радіусами.

- Стійкість до змін освітлення, метод менш чутливий до змін освітлення у порівнянні з деякими іншими методами виявлення об'єктів, оскільки він базується на геометричних ознаках, а не на інтенсивностях пікселів.

- Застосування у реальному часі, завдяки можливості швидкої обробки, метод Hough Circle Transform може бути використаний у системах реального часу, наприклад, у робототехніці, контролі якості та відеоспостереженні.

4.2 Реалізація методу Хафоваго перетворення для колів

Одним з підходів до аналізу зображень є виявлення крапель води на поверхні об'єктиву камери. Ця проблема має практичне значення в різних областях, таких як системи автоматичного водіння, де чистота лінзи впливає на

якість зображення. Метод перетворення Хафа для кіл (Hough Circle Transform) виявився ефективним для вирішення цього завдання.

Для виявлення кіл на зображенні використовується алгоритм, що складається з наступних етапів:

1. Передобробка Зображення:

Завантажене зображення спочатку перетворюється в градації сірого для зменшення обчислювальної складності. Потім воно піддається розмиттю за допомогою гаусового фільтра для видалення шумів і слабких контурів, що можуть вплинути на результати виявлення кіл.

2. Перетворення Хафа для Кіл:

Використовується функція `HoughCircles` із бібліотеки OpenCV, яка реалізує метод перетворення Хафа для виявлення кіл. Параметри цієї функції дозволяють знаходити кількість потенційних крапель. `dp` - інверсне співвідношення роздільної здатності, `minDist` - мінімальна відстань між центрами виявлених кіл, `param1` і `param2` - порогові значення для етапів виявлення кантів і центру кіл відповідно, `minRadius` і `maxRadius` - діапазон розмірів шуканих кіл.

3. Аналіз та Візуалізація Результатів:

Після виявлення кіл, алгоритм аналізує кожне коло, намальовуючи його на зображенні. Визначається загальний рівень забруднення об'єктиву, базуючись на відношенні площі виявлених кіл до загальної площі зображення. Якщо цей рівень перевищує заданий поріг, виводиться повідомлення про забруднення об'єктиву.

Реалізація алгоритму була здійснена на мові програмування Python з використанням бібліотеки OpenCV, код програми надано в Додатку Д. Основна частина програми включає функцію `optimize_drop_detection`, яка приймає шлях до зображення та поріг забруднення як параметри. Ця функція відповідає за виконання усіх вищезазначених етапів та вивід результатів.

Протестуємо та проаналізуємо написаний алгоритм на деяких зображеннях:

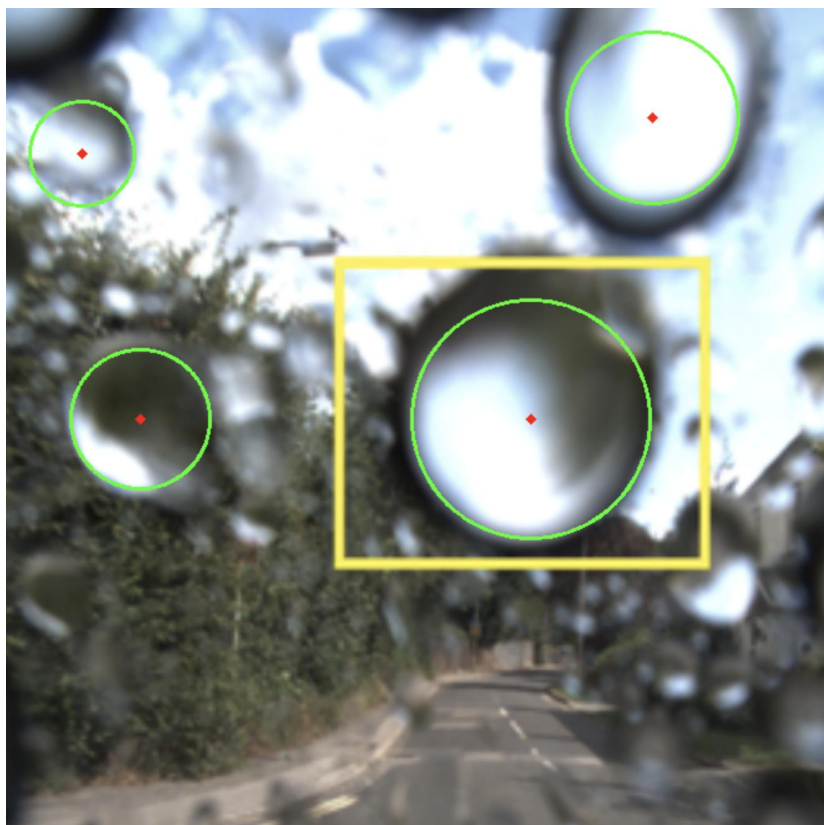


Рисунок 4.1 – Зображення для тесту №1

Як видно з Рисунку 4.1 , метод чітко виявляє та помічає наявні краплі води для візуального розуміння , також виводить у консоль повідомлення - “На об’єктиві присутні краплі.” , яке в подальшому можна використовувати для сповіщення системи. Розглянемо інше зображення та як алгоритм спрацює в

цьому випадку:



Рисунок 4.2 – Зображення для тесту №2

На Рисунку 4.2 показано як відпрацював алгоритм зображення без явного забруднення крапель води, також повідомивши сповіщенням - “Об’єктив чистий.”

4.3 Висновки

Метод перетворення Хафа для кіл виявився потужним інструментом для виявлення крапель води на об’єктиві. Він дозволяє автоматично виявляти та визначати наявність крапель на об’єктиві, що може бути корисним для

багатьох застосувань. Алгоритм був протестований на зображеннях з явним забрудненням і чистим об'єктивом. Результати показали точне виявлення крапель води і без зайвих спрацювань, що підтверджує ефективність методу. Також даний алгоритм можна підлаштовувати під різні задачі, корегуючі параметри крапель які він буде детектувати.

5 МЕТОД ДЕТЕКЦІЇ НАКРИТТЯ ОБ'ЄКТИВУ

Вступ

Накриття об'єктиву камери є значною проблемою для систем відеоспостереження, які залежать від неперервності та якості візуального спостереження. Накриття може виникати через навмисне закривання об'єктиву, накопичення бруду або снігу, а також через помилкове закривання іншими предметами, як-от листям або павутиною. Незалежно від причини, це може призвести до значних безпекових ризиків, оскільки втрачається здатність системи забезпечувати достовірну візуальну інформацію.

Важливість детекції накривання об'єктиву:

1. **Забезпечення безпеки:** Найважливіша роль систем відеонагляду — забезпечення безпеки визначених зон. Накриття об'єктиву може унеможливити виявлення несанкціонованих дій або інших важливих подій.
2. **Запобігання втратам:** В роздрібній торгівлі камери спостереження використовуються для запобігання крадіжкам. Накритий об'єктив камери може призвести до втрат товарів і збитків для бізнесу.
3. **Ефективність реагування:** У випадках надзвичайних ситуацій, таких як пожежі чи медичні надзвичайні ситуації, оперативне спостереження може бути критично важливим для швидкого реагування та надання допомоги.

У цьому розділі ми реалізуємо та протестуємо метод для виявлення та реагування на випадки накриття об'єктиву камер. Перевіримо його на практиці для реальних зображень і оцінемо його коректність. Це допоможе забезпечити більшу ефективність систем відеоспостереження і підвищити рівень безпеки в охоронюваних зонах.

5.1 Методика алгоритму для детекції накриття об'єктиву

Алгоритм, який ми розробляємо для детекції накриття об'єктиву камери, включає послідовність таких кроків, які ефективно визначають і аналізують потенційні затулення [12]. За такими кроками :

1. Конвертація зображення в градації сірого
2. Порогова обробка (Thresholding)
3. Пошук контурів (Finding Contours)
4. Аналіз відсотка затулення
5. Візуалізація

Ця методика надає засоби для автоматичної ідентифікації та аналізу затулення об'єктиву, що є важливим для підтримання функціональності та надійності систем відеонагляду. Тому зараз пройдемося по кожному пункту цього методу та опишемо його.

5.1.1 Конвертація зображення в градації сірого

Конвертація кольорових зображень в градації сірого є критично важливим кроком в багатьох алгоритмах обробки зображень, включаючи детекцію накриття об'єктиву. Перехід до градацій сірого дозволяє знизити складність обчислень, спростити наступні кроки аналізу та зменшити обсяг даних без істотної втрати інформативності.

Ось так виглядає процес конвертації зображення в градації сірого:

1. Вхідні дані:

Зображення в кольоровому просторі RGB (червоний, зелений, синій) містить три канали кольору, кожен з яких несе інформацію про інтенсивність відповідного кольору в кожному пікселі.

2. Формула перетворення:

Щоб перетворити кольорове зображення в зображення в градаціях сірого, використовується вагова сума інтенсивностей кольорових каналів. Формула, яка часто застосовується, виглядає наступним чином:

$$I_{\text{gray}} = 0,299 \times R + 0,587 \times G + 0,114 \times B \quad (18)$$

Коефіцієнти в цій формулі вибрані таким чином, щоб відобразити різну чутливість людського ока до різних кольорів: зелений колір сприймається яскравіше, тому має вищий коефіцієнт, а синій — найменший, оскільки він сприймається темнішим.

3. Обробка кожного пікселя:

Для кожного пікселя кольорового зображення вираховується значення градації сірого за допомогою зазначеної вище формули. Результатом є нове зображення, де кожен піксель представлений одним числовим значенням (від 0 до 255), що відображає яскравість пікселя [13].

Ось які переваги ми отримуємо при конвертації зображення в градації сірого:

- Зменшення обсягу даних: Оскільки зображення в градаціях сірого містить лише один канал порівняно з трьома в RGB, обсяг даних, який потрібно обробляти, зменшується втричі.
- Спрощення алгоритмів: Багато алгоритмів обробки зображень працюють ефективніше на зображеннях в градаціях сірого, оскільки вони вимагають обробки меншої кількості даних.
- Використання в різних застосуваннях: Конвертація в градації сірого є поширеною практикою в багатьох областях, включаючи відеонагляд, медичну візуалізацію, а також в машинному зорі.

Цей крок є фундаментальним для подальшого ефективного виявлення накриття об'єктива та аналізу зображень в контексті безпеки та відеоспостереження.

5.1.2 Порогова обробка (Thresholding)

Порогова обробка зображення є одним з основних методів бінаризації зображень, що використовується для подальшого аналізу і детекції об'єктів у системах відеонагляду. Цей метод дозволяє відділити об'єкти від фону, спрощуючи процес аналізу затулення об'єктива камери.

Процес порогової обробки

1. Визначення порогового значення (T):

Порогове значення вибирається на основі характеристик зображення та конкретного завдання. Воно може бути фіксованим чи динамічно адаптованим в залежності від умов освітлення або інших факторів зображення.

2. Формула бінаризації:

Де кожен піксель зображення в градаціях сірого порівнюється з пороговим значенням (12). Пікселі, що перевищують поріг, встановлюються як 1 (білі), а ті, що нижче порогу, стають 0 (чорні).

3. Результат бінаризації:

Результатом є бінарне зображення, де об'єкти і затулення виділені білим кольором на чорному фоні, що спрощує виявлення та аналіз форм та областей.

Ось які переваги ми отримуємо при пороговій обробці зображення:

- Спрощення зображення: Бінаризація зводить складність візуальних даних, роблячи алгоритми детекції ефективнішими у визначенні форм і меж.

- Підвищення контрастності: Виділення об'єктів від фону полегшує виявлення контурів та інших візуальних ознак.

- Гнучкість застосування: Метод може адаптуватися до різних умов освітлення та типів зображень за допомогою адаптивного порогування.

У контексті детекції накриття об'єктива камери, порогова обробка використовується для ідентифікації раптових змін у візуальних характеристиках зображення, які можуть вказувати на затулення. Наприклад, значне зниження інтенсивності світла на зображенні може бути виявлено як затулення після застосування порогової обробки. Такий підхід дозволяє системі відеонагляду швидко реагувати на потенційні затулення, підвищуючи загальну надійність та ефективність системи безпеки.

5.1.3 Пошук контурів (Finding Contours)

Пошук контурів є ключовою технікою в обробці зображень, що дозволяє ідентифікувати форми та об'єкти на бінарному зображенні, яке було отримане після порогової обробки [14]. Цей процес важливий для детекції накриття об'єктива, оскільки дозволяє точно визначити локацію та масштаби потенційних затулень.

Як виглядає процес пошуку контурів:

1. Визначення контуру - контур в бінарному зображенні визначається як послідовність точок, які утворюють замкнену лінію і відокремлюють білі

та чорні області. Такі контури можуть представляти краї об'єктів, які виділені під час бінаризації.

2. Алгоритми пошуку контурів - популярним методом для знаходження контурів є алгоритм, що використовується в бібліотеці OpenCV, зокрема `'findContours'`. Цей алгоритм аналізує бінарне зображення та виділяє контури на основі змін між чорними та білими областями, ось приклад запису для роботи алгоритму пошуку на мові програмування Python:

`contours, hierarchy = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`, де `'cv2.RETR_EXTERNAL'` вказує на метод вибірки контурів (тільки зовнішні контури), а `'cv2.CHAIN_APPROX_SIMPLE'` зменшує кількість збережених точок на контурі.

3. Обробка та аналіз контурів:

Отримані контури можуть бути аналізовані за допомогою різних геометричних характеристик: площа, периметр, описуючий прямокутник, центр мас тощо. Це дозволяє оцінити розміри та форму областей затулення.

`area=cv2.contourArea(contour)`

`x, y, w, h=cv2.boundingRect(contour)`

Значення площі та відношення розмірів можуть використовуватись для ідентифікації та класифікації типів затулення.

У контексті детекції накриття об'єктиву, знання точного розташування та форми затулення дозволяє системі робити інформовані висновки про потенційні загрози безпеці або технічні проблеми. Визначення контурів затулення може також сприяти автоматизації процесу реагування на інциденти,

наприклад, шляхом активації сигналізації або сповіщення операторів системи безпеки.

Візуалізація контурів на зображенні може виконуватися за допомогою функцій OpenCV, як-от `cv2.drawContours`, що дозволяє користувачам легко ідентифікувати проблемні ділянки і оцінити ефективність алгоритму детекції.

5.1.4 Аналіз відсотка затулення

Аналіз відсотка затулення є критично важливим кроком у процесі детекції накриття об'єктиву, оскільки дозволяє кількісно оцінити ступінь затулення зображення. Цей аналіз допомагає визначити, чи перевищує затулення певний поріг, що може вимагати втручання або подальшого розслідування.

Кроки процесу аналізу відсотка затулення

1. Обчислення площі контурів - після ідентифікації контурів на бінарному зображенні, необхідно обчислити площу кожного контуру, яка вказує на розмір затулених областей. Це може бути зроблено за допомогою функції, наприклад `cv2.contourArea` у бібліотеці OpenCV.

$$\text{Area_of_contour} = \text{cv2.contourArea}(\text{contour})$$

2. Обчислення загальної площі затулених областей - сумування площ усіх контурів дає загальну площу затулення на зображенні :

$$\text{total_area_of_contours} = \sum_{i=1}^n \text{area_of_contours} \quad (19)$$

, де n — кількість ідентифікованих контурів.

3. Обчислення відсотка затулення - відсоток затулення обчислюється як відношення загальної площі затулених областей до площі всього зображення.

$$\text{Coverage} = \left(\frac{\text{total_area_of_contours}}{\text{total_image_area}} \right) \times 100\% \quad (20)$$

, де total_image_area — площа всього зображення, яка визначається як добуток ширини та висоти зображення.

4. Визначення порогу затулення - залежно від специфікацій системи безпеки, встановлюється поріг відсотка затулення, при перевищенні якого система сприймає затулення як критичне. Наприклад, поріг може бути встановлений на рівні 60%.

5. Реагування на затулення - якщо обчислений відсоток затулення перевищує встановлений поріг, система може активувати сповіщення або інші заходи реагування, такі як зміна кута зору камери, активація додаткових камер або відправлення сповіщення оператору системи.

Застосування аналізу відсотка затулення - цей аналіз є особливо важливим у високочутливих застосуваннях, таких як системи безпеки в аеропортах, банках або урядових установах, де важливо швидко реагувати на будь-які спроби заблокувати або саботувати камери. Він також корисний в інтелектуальних транспортних системах, де затулення камер може вплинути на безпеку руху.

Під час аналізу відсотка затулення можуть використовуватися візуальні індикатори на зображенні для позначення зон, які виявлені як затулені. Це дозволяє операторам системи швидко визначити і оцінити масштаби проблеми

та вжити відповідних заходів.

5.1.5 Візуалізація

Візуалізація затулених областей на зображенні є важливим етапом у процесі детекції накриття об'єктиву камери. Це не тільки допомагає в оперативному виявленні і діагностиці проблем, але й забезпечує зручність у сприйнятті результатів аналізу людьми, які здійснюють моніторинг систем безпеки.

1. Виділення затулених областей - застосування графічних методів для підсвічування або обрамлення областей на зображенні, де було виявлено затулення. Це може бути реалізовано за допомогою різних графічних технік, таких як зміна кольору контурів або додавання інформативних написів [15].
2. Маркування контурів - з використанням функцій бібліотеки обробки зображень, наприклад OpenCV, можна маркувати контури затулених областей на оригінальному або обробленому зображенні.

$$\text{cv2.drawContours}(I_{\text{original}}, \text{contours}, -1, (0,255,0), 3) \quad (21)$$

, де `contours` — це список контурів, отриманих на попередніх етапах, а `(0,255,0)` представляє колір (зелений), в якому контури будуть відображені.

Основні переваги при застосуванні візуалізації:

- Підвищення інформативності: Візуалізація дозволяє операторам системи безпеки швидше зорієнтуватись у ситуації, оцінити ступінь та специфіку затулення.

- Архівація та звітність: Візуалізовані зображення можуть бути збережені для подальшого аналізу або як доказ у разі розслідувань.

- Тренування та калібрування: Візуалізація також може бути використана для тренування та калібрування алгоритмів детекції, надаючи зворотний зв'язок щодо їхньої ефективності.

Потужні бібліотеки обробки зображень, такі як OpenCV, надають широкий спектр функцій для візуалізації, що дозволяє легко інтегрувати цей процес у більшість систем відеоспостереження. Інтеграція візуалізації у програмне забезпечення забезпечує не тільки зручність у використанні, але й підвищує загальну надійність та реактивність систем безпеки.

5.2 Реалізація та тестування методу

За методикою яка була описана вище - реалізуємо цей алгоритм детекції затулення об'єктиву камери відео спостереження в середовищі PyCharm, вихідний код цього методу буде наведено в Додатку Е.

Тож перейдемо до безпосереднього тесту нашого алгоритму на конкретних зображеннях:

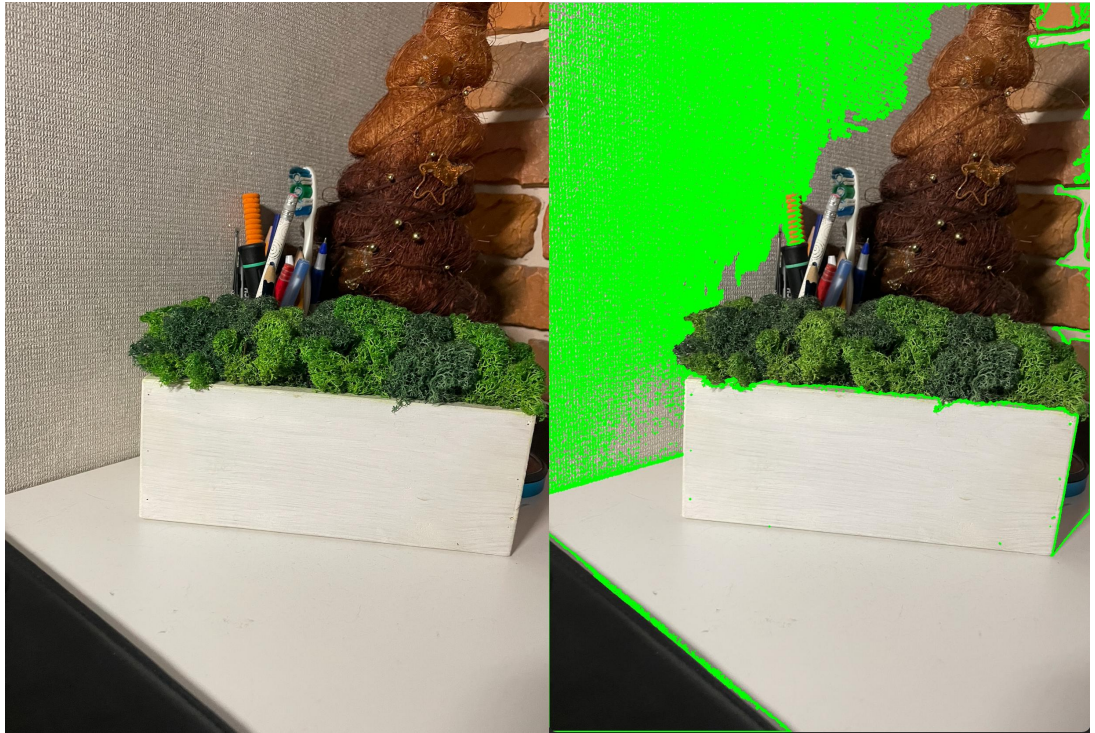


Рисунок 5.1 – Тестове зображення затулення 1

На данному Рисунок можна побачити , як відпрацьовує алгоритм по детекції затулення об’єктиву і наглядно нам вказує на ймовірні області затулення, за допомогою вибору порогового значення відсотка забруднення можна налаштувати, коли програмі вказувати що об’єктив затулено, в нашому випадку висталено поріг на 50% для того щоб знизити кількість хибних спрацювань. Також в консолі PyCharm виводиться такий текст - “Камера не затулена 37.259569119868424”, тут ми отримуємо результат забруднена камера чи ні, і відповідно відсоток цього забруднення, який склав порядку 37%.

Тепер порівняємо друге зображення:

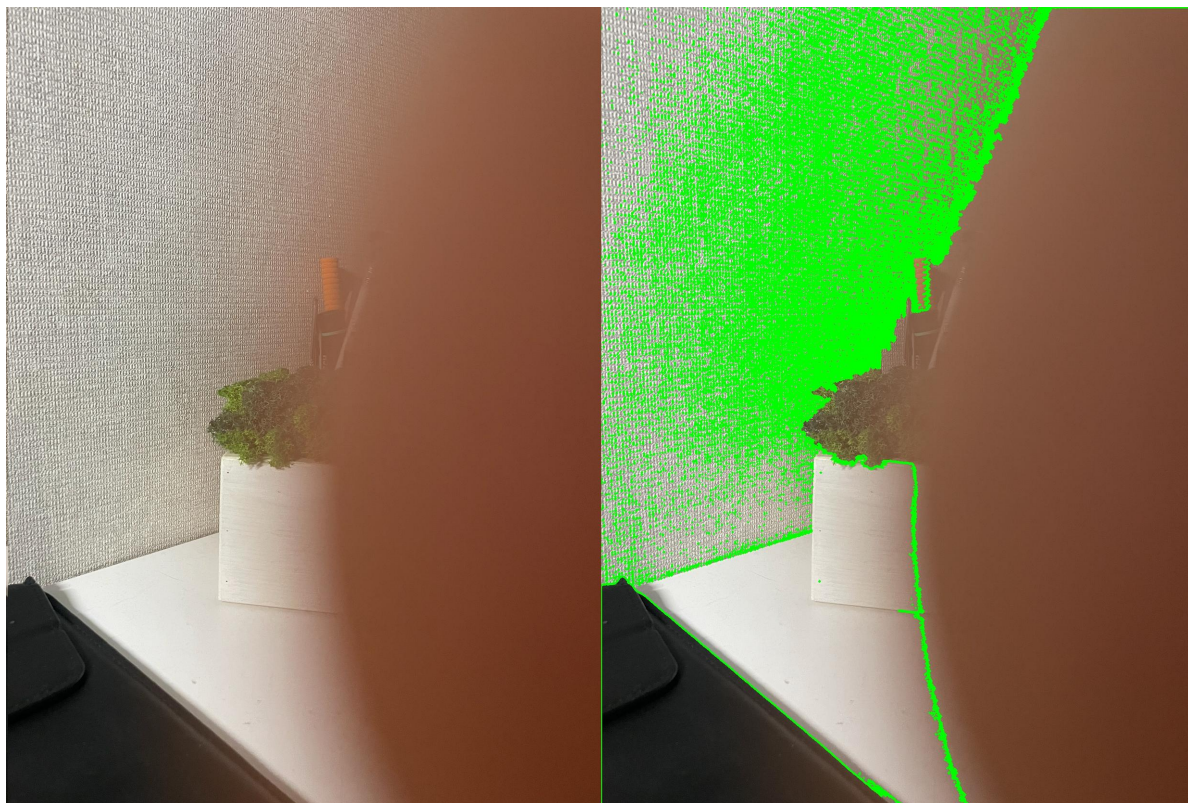


Рисунок 5.2 – Тестовве зображення затулення 2

Як видно з Рисунку 5.2 - об'єктив явно затулений об'єктом і так само алгоритм візуалізує ймовірну область затулення. Результат який ми отримуємо в консолі - "Камера затулена 51.984806723667745" - отримали повідомлення що наша камера затулена і затулення складає майже 52%.

Розглянемо ще 2 зображення та представимо у наступних рисунках:

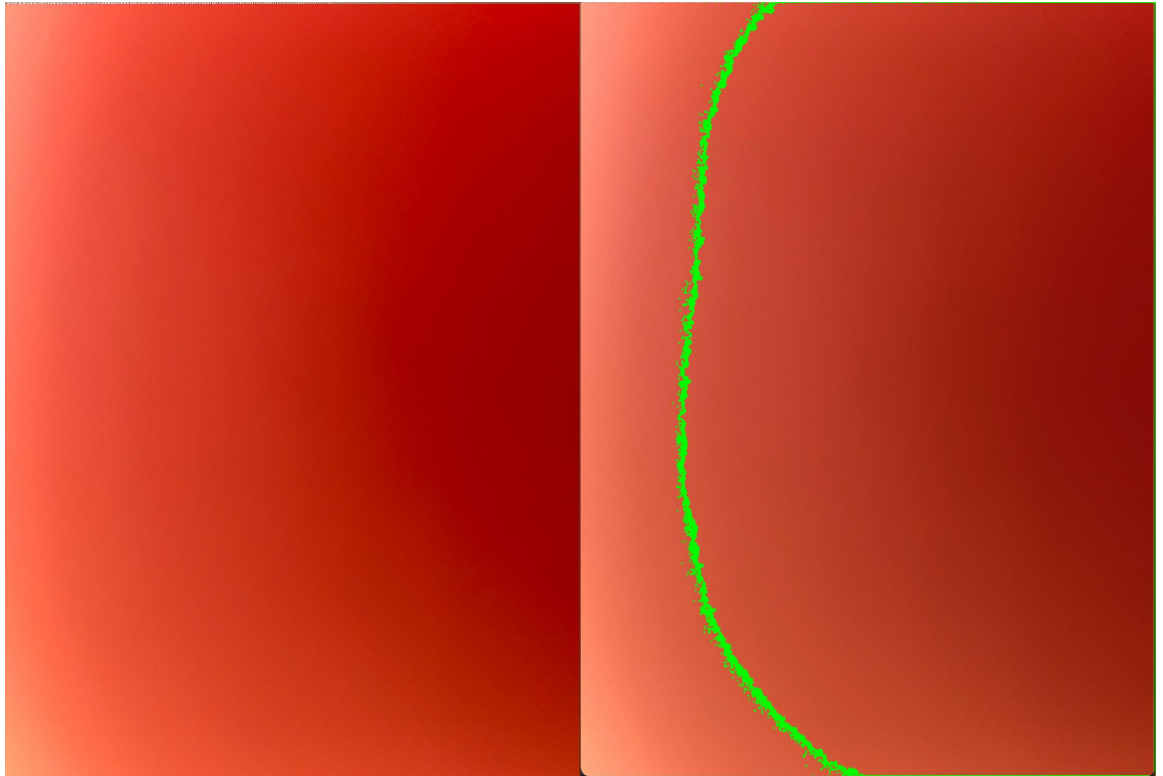


Рисунок 5.3 – Тестове зображення затулення 3

Дане зображення було зроблено на тому самому місці із перекритим об'єктивом камери, з цього рисунка видно що даний алгоритм візуально показав з правої сторони перекриття, що стосується повідомлення яке виводиться - "Камера затулена 76.28971509837936", і дає нам знати що камера затулена. Відповідна ситуація стосується і наступного зображення - Рисунок 5.3, на ньому явно видно суцільно закритий об'єктив камери пальцем, і в результаті ми отримуємо повідомлення про затулення - "Камера затулена 99.86032479141835" на 99.9%. відповідно можна зробити висновок про коректність роботи реалізованого алгоритму.

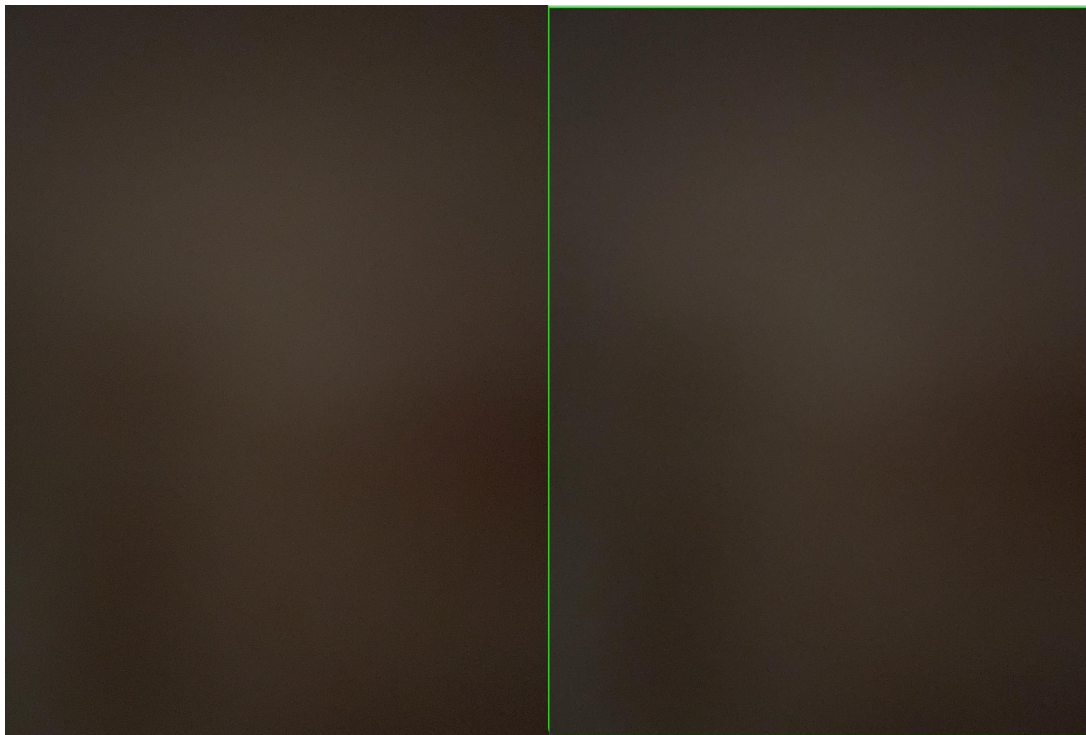


Рисунок 5.4 – Тестове зображення затулення 4

5.3 Висновки

В данному розділі вдалося реалізувати алгоритм по детекції затулення об'єктиву камери, в наведених рисунках було наглядно продемонстровано, як відпрацьовує алгоритм та візуалізує ймовірні області затулення. Також на виході ми отримуємо повідомлення про статус камери і відсоток ймовірного затулення, поріг якого можна за необхідністю корегувати під власті задачі.

6 МЕТОД ДЕТЕКЦІЇ ЗАСВІТЛЕННЯ ОБ'ЄКТИВУ

Вступ

Засвітлення об'єктиву, відоме також як лінзове засліплення або світловий виблиск, є значною проблемою в системах відеоспостереження, яка може суттєво вплинути на якість зображення та надійність системи. Це явище виникає, коли інтенсивне світло потрапляє прямо в об'єктив камери, спричиняючи розмиття, зниження контрасту та виникнення розсіяних світлових плям, що ускладнює ідентифікацію осіб або об'єктів на зображенні [16].

Визначення та коригування засвітлення об'єктиву є критично важливим для підтримки оптимальної роботи систем відеоспостереження, особливо в умовах різкої зміни освітлення, таких як пряме сонячне світло або високі відблиски від водних та металевих поверхонь. Належне виявлення засвітлення дозволяє оперативно адаптувати параметри камери для забезпечення чіткості зображення.

Методи детекції засвітлення об'єктиву зазвичай зосереджуються на аналізі відповідних змін у світловій інтенсивності та кольорових розподілах зображення. До ключових підходів відносяться використання алгоритмів машинного навчання для класифікації зображень на основі характерних ознак засвітлення, аналіз частотних характеристик та реалізація методів динамічного коригування налаштувань камери в реальному часі.

У цьому розділі ми розглянемо практичну реалізацію алгоритму, який буде використовуватись для ідентифікації та коригування засвітлення об'єктиву. Алгоритм зосереджений на обробці вхідних зображень для визначення аномальних світлових умов, що впливають на якість візуальної інформації. Він буде інтегрувати передові методи обробки зображень та аналізу даних для виявлення характерних ознак засвітлення, що дозволить системі автоматично адаптуватися та оптимізувати умови зйомки, забезпечуючи високу якість відеоматеріалів незалежно від зовнішніх умов освітлення [17].

6.1 Методика та реалізація алгоритму для детекції засвітлення об'єктиву

Для створення алгоритму детекції засвітлення об'єктиву камери використаємо аналогічну методику - яку було описано в розділі 5, яка допомагає систематизувати підхід і забезпечити точність та ефективність рішення:

- Завантаження зображення: Імпорт зображення, з яким буде проводитися робота.
- Перетворення зображення: Конвертація у відтінки сірого для спрощення подальшої обробки.
- Порогова обробка: Виділення дуже яскравих областей, які можуть свідчити про засвітлення.
- Детекція контурів: Знаходження і аналіз контурів яскравих областей для

визначення їхніх характеристик.

- Розрахунок відсотка засвітлення: Визначення відсоткового співвідношення засвітленої площі до загальної площі зображення.
- Візуалізація результатів: Підсвічування засвітлених областей на зображенні.

Для реалізації задачі, як і в попередніх розділах було вибрано - Python з бібліотекою OpenCV через її великі можливості у обробці зображень та наявність необхідних інструментів для аналізу та візуалізації даних.

Так само по аналогії до попереднього розділу - конвертуємо вхідне зображення в градації сірого. Після цього йде - порогова обробка в якій і криється основна відмінність.

Функція порогової обробки в OpenCV, `cv2.threshold`, використовується для визначення та зміни пікселів зображення на основі порігу яскравості. Дві з найбільш поширених опції порогової обробки — `cv2.THRESH_BINARY` та `cv2.THRESH_BINARY_INV` (яка використовувалася в розділі 5). Ось як вони працюють і в чому їх різниця:

`cv2.THRESH_BINARY`

Цей метод встановлює пікселі зображення на максимальне значення (зазвичай 255), якщо їхнє значення більше за певний поріг. Пікселі, які мають значення нижче порогу, стають нулем, за такою формулою, яка подібна до (12):

$$\text{dst}(x, y) = \begin{cases} \text{maxValue} & \text{if src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \quad (22)$$

, де ``src(x, y)`` — значення пікселя в оригінальному зображенні, ``thresh`` — порогове значення, ``maxValue`` — максимальне значення, яке зазвичай встановлюється на 255 для 8-бітних зображень.

``cv2.THRESH_BINARY_INV``

Цей метод працює протилежно до ``cv2.THRESH_BINARY``. Він встановлює значення пікселів на нуль, якщо вони перевищують певний поріг, і на максимальне значення, якщо їхнє значення менше порогу. Формула така:

$$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxValue} & \text{otherwise} \end{cases} \quad (23)$$

Як і було сказано ці два методи схожі, але використовуються для різних цілей, ``cv2.THRESH_BINARY`` може бути корисним для виділення світлих об'єктів на темному фоні, тоді як ``cv2.THRESH_BINARY_INV`` — для виділення темних об'єктів на світлому фоні.

Вибір між цими методами залежить від специфічних потреб задачі, яка стоїть перед нами. Відповідно в цьому розділі нас цікавить детекція світлих участків на зображення, які потенційно можуть бути засвітленням об'єктиву - тому замінемо функцію порогової обробки на - `cv2.THRESH_BINARY`, в робочій програмі наданій в Додатку Е.

Пошук контурів залишається таким самим, як і було описано в пункті 5.1.3. Те саме можна сказати і про аналіз відсотку засвітлення та візуалізації, описані в пунктах 5.1.4 та 5.1.5 відповідно, вони залишаються без змін.

6.2 Тестування методу детекції засвітлення об'єктиву

Для цього тесту було підготовлені зображення для перевірки роботи реалізованого алгоритму, вихідний код якого наведено в Додатку Є, перейдемо для тесту та аналізу зображень:

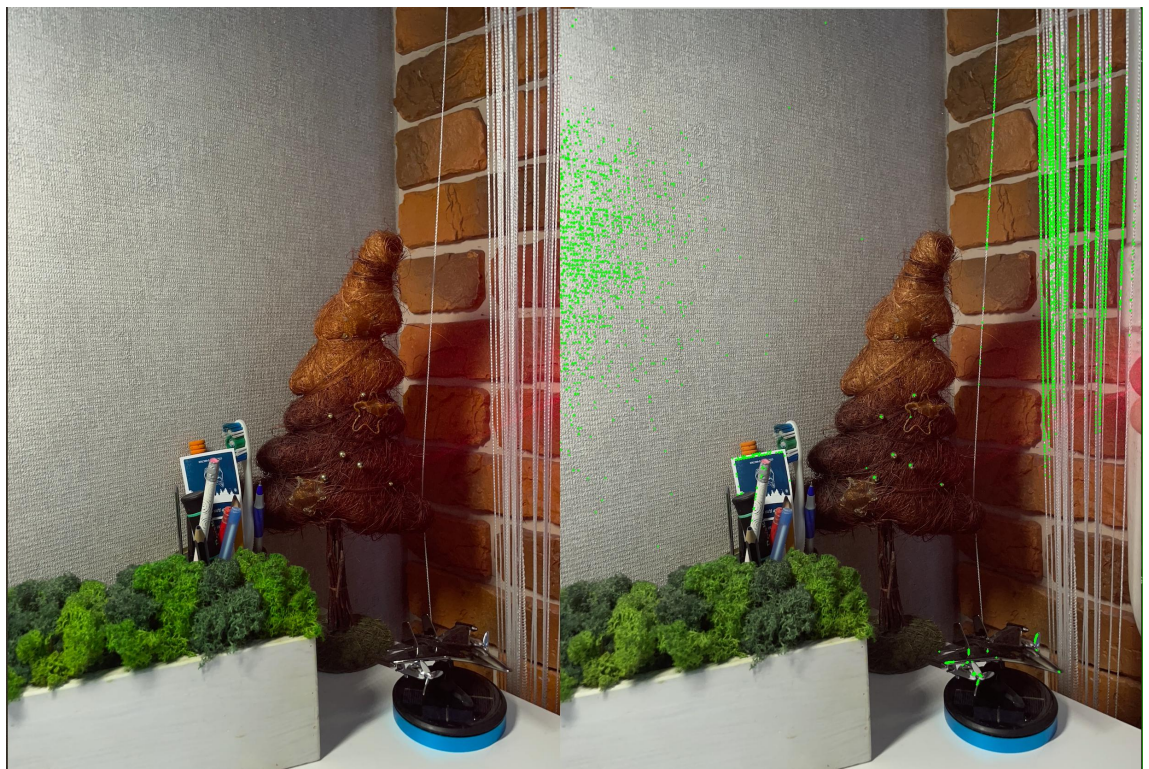


Рисунок 6.1 – Тестове зображення засвітлення 1

На рисунку 6.1 ми маємо дві версії фото: зліва оригінальне зображення, а праворуч - оброблене, де зеленими точками виділені потенційно засвітлені ділянки.

На обробленій частині зображення (праворуч) видно численні зелені точки. Ці точки, були ідентифіковані як засвітлені ділянки. Це можуть бути області, де інтенсивність світла перевищує заданий поріг, але, враховуючи невеликий

відсоток (близько 0.064%), ці засвітлення не є значними.

Консольне повідомлення вказує на те, що засвітлення становить приблизно 0.064% від загальної площі зображення, і на основі цього алгоритм робить висновок, що "Об'єкти не засвітлено." Алгоритм коректно ідентифікував та візуалізував потенційні області засвітлення, але визнав їх несуттєвими для визначення загального стану засвітлення камери.

Перейдемо до наступного тестового зображення:

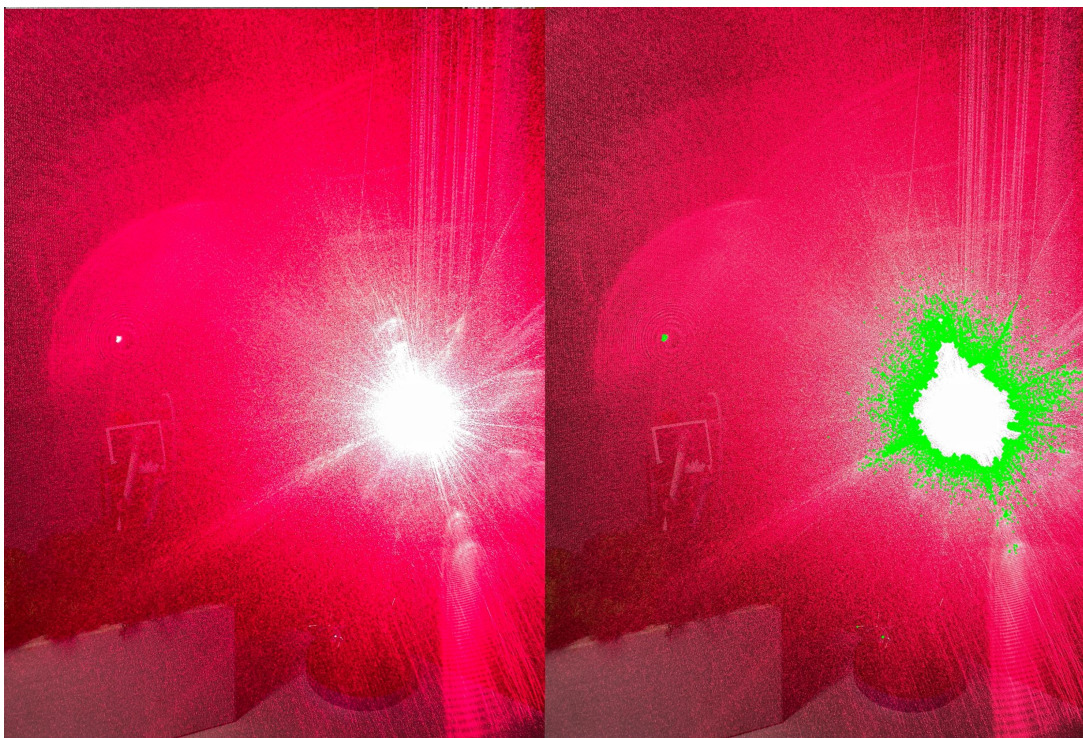


Рисунок 6.2 – Тестове зображення засвітлення 2

На Рисунку 6.2, також видно дві версії: зліва оригінальне зображення, де яскраве світло створює значне засвітлення (за допомогою направленою лазера), і праворуч оброблене зображення, де засвітлені ділянки виділені зеленим кольором.

На обробленій частині зображення (праворуч) велика область засвітлення

зеленого кольору, яка обрамляє надзвичайно яскраве джерело світла. Це вказує на те, що алгоритм ефективно ідентифікував області з високою інтенсивністю світла, які значно впливають на якість зображення.

Консольне повідомлення вказує, що засвітлення становить близько 2.28% від загальної площі зображення, і на основі цього алгоритм визначив, що "Об'єктив засвітлено." Відсоток засвітлення значно більший порівняно з попереднім зображенням, і він достатній для класифікації як суттєве засвітлення.

Цей аналіз демонструє, що алгоритм здатен ефективно виявити і відобразити засвітлення на зображенні, дозволяючи визначити його вплив на якість зображення та класифікувати стан об'єктиву відповідно до визначеного порогу значущості.

І проаналізуємо роботу алгоритму на останньому зображенні:



Рисунок 6.3 – Тестове зображення засвітлення 3

На рисунку 6.3 по аналогії до попередніх рисунків є оригінальне зображення (зліва) та оброблене (праворуч), де виділена область засвітлення зеленим кольором.

На обробленій частині зображення видно, що дуже яскраве світло від світильника було визначено як значне засвітлення. Область, що оточує світлове джерело, чітко обведено зеленою лінією, що вказує на велику інтенсивність та розмір засвітлення. Консольне повідомлення показує, що засвітлення становить близько 11.8% від загальної площі зображення. Відсоток значно вищий, порівнюючи з попередніми випадками, і на основі цього алгоритм визначив, що "Об'єктив засвітлено."

6.3 Висновки

В цьому розділі, вдалося реалізувати метод детекції засвітлення об'єктиву камери, було досліджено його ефективність у виявленні та візуалізації засвітлених областей на зображеннях. Алгоритм продемонстрував високу точність у визначенні областей з високою інтенсивністю світла, що ілюструє його практичну застосовність у контекстах, де необхідне автоматизоване рішення для оцінки якості зображення.

Алгоритм використовує порогову обробку для ідентифікації світлих ділянок та обчислює їхній відсоток від загальної площі зображення, що дозволяє кількісно оцінити ступінь засвітлення. Візуалізація засвітлених областей

забезпечує наглядність результатів і може слугувати корисним інструментом для подальшого аналізу.

Однією з ключових переваг алгоритму є його гнучкість: можливість налаштування порогу детекції та чутливості до яскравості дозволяє адаптувати метод до різних умов зйомки та специфічних вимог.

7 РЕАЛІЗАЦІЯ АЛГОРИТМУ ОЦІНКИ ЯКОСТІ ЗОБРАЖЕННЯ

Фінальний розділ присвячений реалізації комплексного алгоритму оцінки якості зображення, який об'єднує всі попередні методи оцінки забрудненості об'єктиву. Ключова мета даного розділу полягає у створенні інтегрованої системи детекції різних видів артефактів та дефектів зображення, що забезпечує надійність і точність виявлення порушень роботи системи відеоспостереження.

У попередніх розділах були детально розглянуті та реалізовані окремі методи оцінки якості зображення, які дозволяють визначати основні проблеми, пов'язані з порушенням роботи камер відеоспостереження:

- Розділ 2: Метод детекції порушення фокусування: у цьому розділі був представлений алгоритм, спрямований на виявлення та оцінку дефокусування камери. Цей метод використовує аналіз різкості зображення для виявлення відхилень від оптимального фокусування.
- Розділ 3: Метод детекції забруднення об'єктиву: розділ містить детальний опис алгоритму, який дозволяє виявляти забруднення об'єктиву шляхом аналізу аномалій в інтенсивності пікселів на зображенні та візуального виявлення патернів забруднення.
- Розділ 4: Метод детекції крапель води на об'єктиві: у цьому розділі розроблений алгоритм для визначення присутності крапель води на об'єктиві камери шляхом аналізу форми та розподілу світлових плям, які

виникають у результаті заломлення світла.

- Розділ 5: Метод детекції накриття об'єктиву: методика цього розділу ґрунтується на аналізі змін яскравості зображення та пошуку характерних патернів, які вказують на повне або часткове блокування об'єктиву.

- Розділ 6: Метод детекції засвітлення об'єктиву: у шостому розділі описаний алгоритм, спрямований на виявлення засвітлення об'єктиву шляхом аналізу розподілу світла та інтенсивності кольору на зображенні.

Об'єднання усіх методів в один комплексний алгоритм дозволить автоматично визначати якість зображення в режимі реального часу та оперативно виявляти різні види дефектів та артефактів. Ключові етапи реалізації алгоритму:

1. Вивід вхідного зображення;
2. Аналіз фокусування (Додаток А);
3. Детекція забруднення об'єктиву (Додаток Г);
4. Виявлення крапель води (Додаток Д);
5. Аналіз накриття об'єктиву (Додаток Е);
6. Детекція засвітлення об'єктиву (Додаток Є).

Відповідно цим ключовим етап і буде сформовано вихідний комплексний алгоритм для оцінки зображення та детекції артефактів на зображеннях. Це дозволить вчасно виявляти порушення роботи системи відеоспостереження та підтримувати оптимальну якість відеоматеріалів. Вихідний код програми наведено в Додатку Ж.

В цій програмі були зроблені деякі корекції такі як:

- Вивід зображення не відбувається на кожному етапі, а лише після аналізу зображення, якщо на ньому присутній який-небудь артефакт - тоді на виводиться зображення для певного етапу з візуалізацією конкретних проблеми.

- В консоль виводиться висновок по кожному етапу аналізу з вихідними значеннями для подальшого аналізу, який був наведений вище, відповідно.

Тепер наглядно протестуємо, як комплексно працює реалізований алгоритм детекції забрудненості об'єктиву. Проаналізуємо вхідне зображення, яке було представлено на Рис.7:

Вхідне зображення

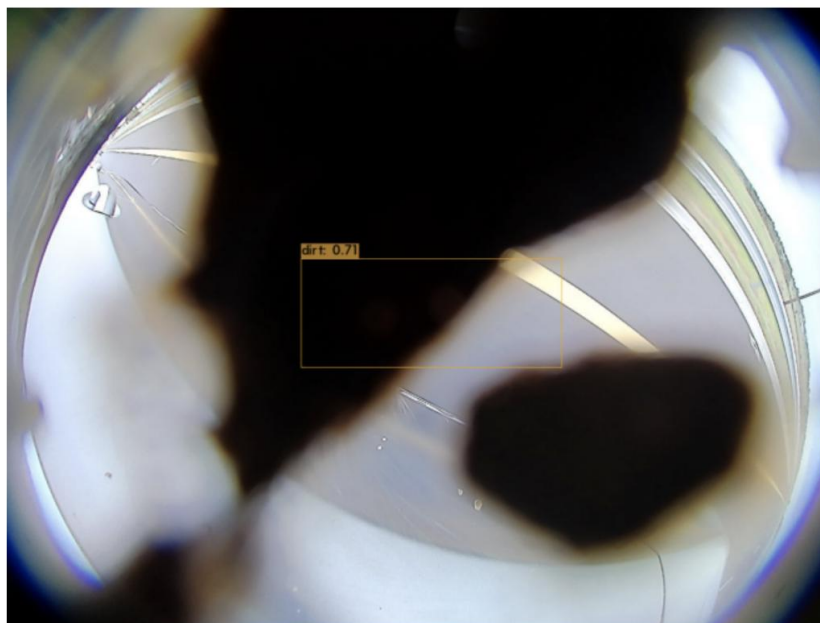


Рисунок 7.1 – Вхідне зображення для тесту

На оригінальному зображенні видно суттєву затуленість, яка, ймовірно, спричинена перешкодою перед об'єктивом, а ось який результат ми отримали

від програми, в повідомлення консолі:

“Висновки після обробки зображення:

Камера не сфокусована: 21.28

Камера забруднена: True, Площа забрудненості: 28.59%

Краплі на об'єктиві відсутні.

Камера затулена, відсоток затулення: 50.66467359676848%

Об'єктив не засвітлено, відсоток засвітлення: 3.981230815751338 %”

В такому форматі виводиться інформація по кожному зображенню, при наявності артефактів (як в нашому випадку) виводяться зображення на відповідних етапах:

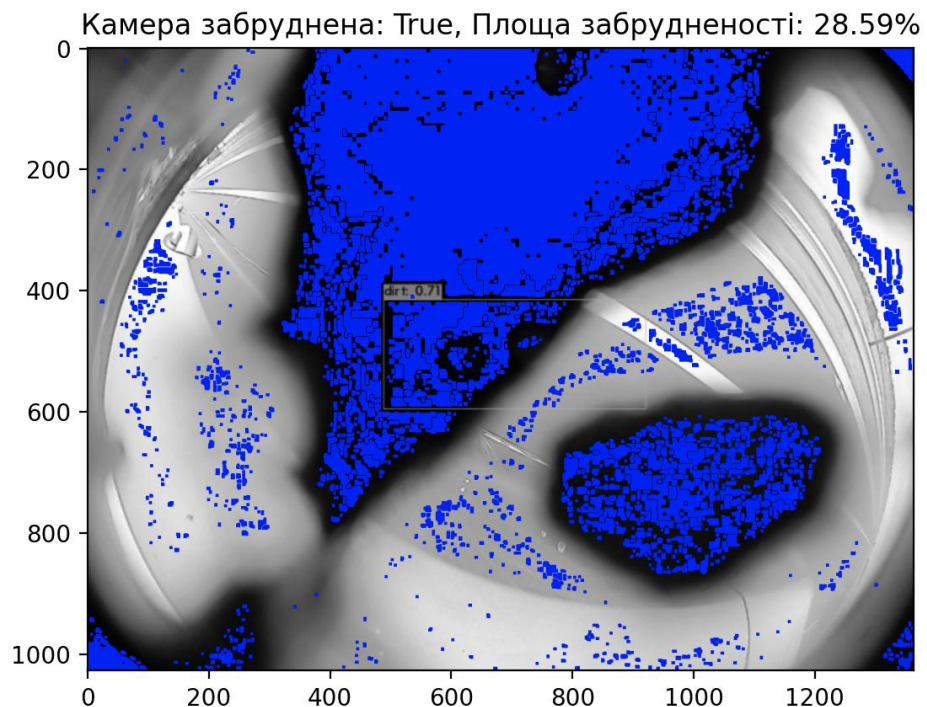


Рисунок 7.2 – Детекція забруднення об'єктиву

На цьому зображенні результатів аналізу забрудненості видно виділені блакитним кольором області забруднення об'єктиву. Додатково включено повідомлення з наступними висновками: “Камера забруднена: Ідентифіковано області, що становлять 28.59% від загальної площі зображення.”

На зображенні результатів аналізу затуленості (Рис.19) видно виділені зеленим контури затулених областей об'єктиву. Додатково включено повідомлення:

Камера затулена: Відсоток затулення становить 50.66%. Камера затулена через перешкоди перед об'єктивом, що призводить до значного перекриття поля зору та погіршення якості зображення.

Камера не сфокусована: Значення розмиття становить 21.28 (протестувавши алгоритм аналізу сфокусованості зображення, поріг при якому погане фокусування становить - 100).

Об'єктив не засвітлено: Відсоток засвітлення становить 3.98%.

Відсутні краплі на об'єктиві.

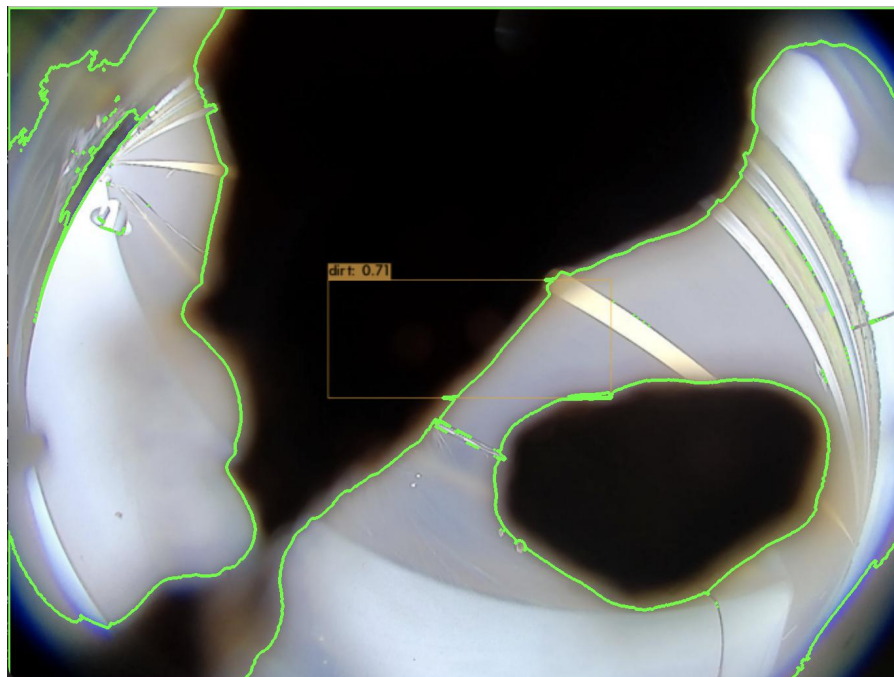


Рисунок 7.3 – Аналіз затулення об'єктиву

Алгоритм успішно провів комплексний аналіз стану об'єктива камери, враховуючи такі критерії, як фокусування, забрудненість, присутність крапель, затуленість та засвітлення. Виявлено, що об'єктив потребує очищення та видалення перешкод, що затуляють поле зору.

Протесстуємо наступне зображення представлене на Рис.7.4:

Вхідне зображення



Рисунок 7.4 – Зображення з краплями води на об'єктиві

Це вхідне зображення з явними краплями на об'єктиві, а аналіз зображення алгоритмом видає ось такий результат:

“Висновки після обробки зображення:

Камера не сфокусована: 14.19

Камера не забруднена, площа забрудненості: 5.41%

На об'єктиві присутні краплі.

Камера не затулена, відсоток затулення: 25.068954627763663%

Об'єктив засвітлено, відсоток засвітлення: 17.912230935482928 %”

На зображенні результатів аналізу детекції крапель (Рисунок 7.5) видно зелені круги навколо крапель води. Додатково включено повідомлення: “На

об'єктиві присутні краплі.”

На зображенні результатів аналізу засвітлення (Рисунок 7.6) видно виділені зеленим контури засвітлених областей. Додатково включено повідомлення: “Об'єktiv засвітлено: Відсоток засвітлення становить 17.91%”

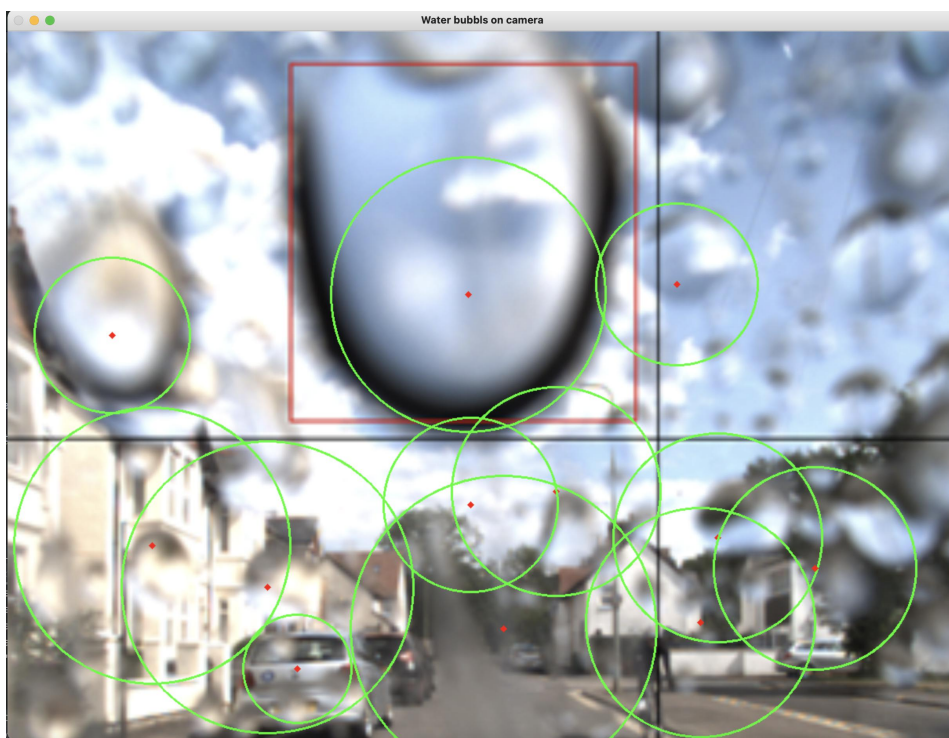


Рисунок 7.5 – Аналіз та детекція крапель води

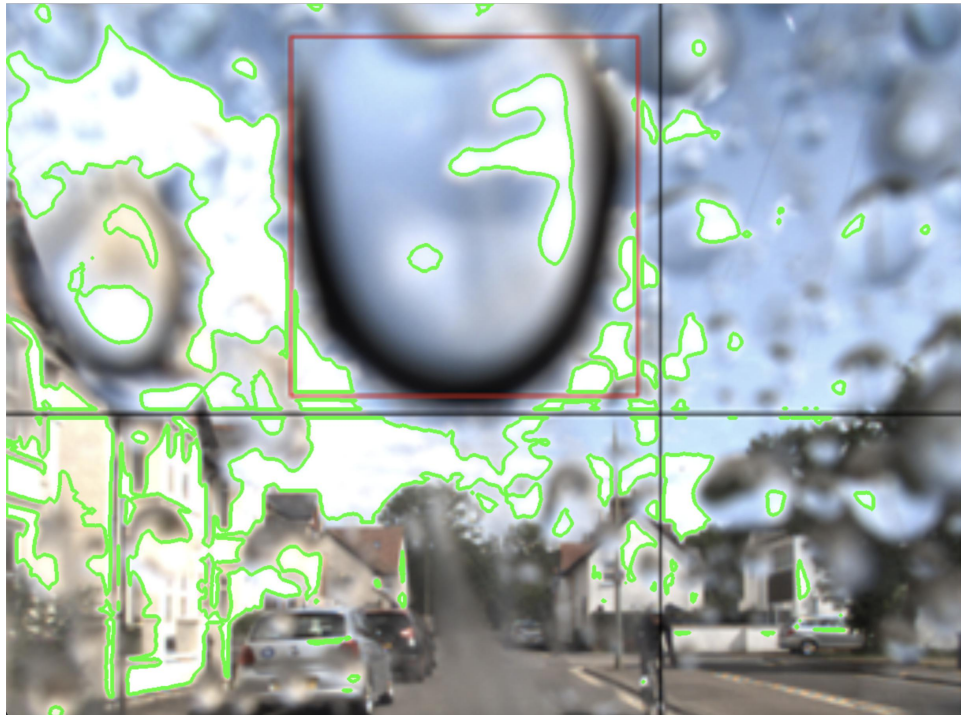


Рисунок 7.6 – Аналіз та детекція засвітленості об'єктиву

Комплексний алгоритм продемонстрував ефективність у виявленні та аналізі різних дефектів, таких як фокусування, забрудненість, затуленість, краплі та засвітлення на об'єктиві камери. Камера не сфокусована, що потребує корекції фокусування. На об'єктиві присутні краплі води, які слід видалити. Камера не критично затулена, але є певні перешкоди в полі зору. Об'єктив засвітлено на 17.91%, що може значно вплинути на якість зображення.

ВИСНОВКИ

У цій магістерській дисертації було розглянуто проблему детекції порушення якості зображення для камер спостереження, а також запропоновано комплексний підхід до вирішення цієї проблеми.

Основна мета роботи полягала у розробці методів детекції порушення фокусування, забруднення об'єктиву, крапель води на об'єктиві, накриття об'єктиву та засвітлення об'єктиву для необслуговуваних або вбудованих камер спостереження з метою сигналізації про проблеми або керування моторизованими засобами підтримки.

Досягнення цілей дослідження

У ході дослідження було реалізовано комплексний алгоритм детекції всіх перелічених проблем якості зображення:

1. Методи аналізу відеозображення

Було розглянуто основні принципи та методи обробки відеозображень для виявлення дефектів та порушень у роботі камер спостереження.

2. Методи детекції порушення фокусування

Розроблено та реалізовано алгоритм для виявлення дефокусування камер, який базується на аналізі різкості зображення.

3. Методи детекції забруднення об'єктиву

Розроблений метод дозволяє виявляти різні види забруднення об'єктиву

шляхом аналізу інтенсивності пікселів та характерних патернів забруднення.

4. Методи детекції крапель води на об'єктиві

Представлений алгоритм детекції крапель води дозволяє визначати специфічні світлові плями, що виникають через заломлення світла на краплях води.

5. Методи детекції накриття (затулення) об'єктиву

Алгоритм забезпечує виявлення повного або часткового блокування об'єктиву на основі аналізу яскравості та контурних патернів.

6. Методи детекції засвітлення об'єктиву

Розроблений алгоритм здатен виявляти засвітлення об'єктиву через лазерне або сонячне світло шляхом аналізу розподілу світла та кольору на зображенні.

Було реалізовано робочий комплексний алгоритм детекції всіх перелічених проблем, який перевіряє зображення за кожним критерієм та дає висновок щодо наявності забруднень або порушень, а також визначає відсоток проблемності для кожного етапу.

Розроблений алгоритм відкриває широкі перспективи для застосування у різних сферах:

- Безпека та відеоспостереження

Система може бути інтегрована у необслуговувані камери спостереження, забезпечуючи автоматичну сигналізацію про проблеми з якістю зображення та керуючи моторизованими засобами підтримки.

- Інтелектуальні транспортні системи

Виявлення проблем з якістю зображення в системах відеоспостереження дозволяє підтримувати безпеку та точність роботи інтелектуальних транспортних систем.

- Промисловий моніторинг

Система може використовуватися для моніторингу технологічних процесів, де надійність та якість зображення є ключовими факторами.

- Дослідницькі та наукові проекти

Використання комплексного алгоритму може допомогти в автоматизації збору даних та підвищенні точності наукових досліджень.

Розроблена система є важливим внеском у підвищення надійності та автоматизації роботи камер спостереження, забезпечуючи ефективний моніторинг та управління у реальному часі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wong, S., & Jassemi-Zargani, R. (2014). Predicting image quality of surveillance sensors. URL: (https://cradpdf.drdc-rddc.gc.ca/PDFS/unc197/p800925_A1b.pdf)
2. Zeng, Y. C., Chueh, M. F., & Tsai, C. H. (2012). Automatic recognition of frame quality degradation. URL: (http://www.apsipa.org/proceedings_2012/papers/94.pdf)
3. Wang, Y. K., et al. (2011). Real-time camera anomaly detection for real-world video surveillance. URL:(https://www.researchgate.net/publication/224258100_Real-time_camera_anomaly_detection_for_real-world_video_surveillance)
4. Rezaei, M., et al. (2021). Detecting and mitigating camera anomalies in vision-based self-driving. *Applied Soft Computing*, 109, 107563.
URL: (<https://www.sciencedirect.com/science/article/abs/pii/S0029801821008404>)
5. George, D. (2011). Python video analysis tools.
URL:(https://books.google.com.ua/books?hl=uk&lr=&id=J9b_CH-NrycC&oi=fnd&pg=PP2&dq=Python+video+analysis+tools&ots=B044P07Ilz&sig=nf4Jyd8la5ErYuWox8C_0zEDsrY&redir_esc=y#v=onepage&q=Python%20video%20analysis%20tools&f=false)
6. Rosebrock, A. (2019). Video analysis with OpenCV and Python. *Journal of Open Source Software*, 4(36), 1800.
URL :(<https://joss.theoj.org/papers/10.21105/joss.01800>)
7. Stelmach, G., et al. (2013). Detecting of Unique Image Features by Using Camera

with Controllable Parameters.

URL:(https://www.researchgate.net/publication/236110827_Detecting_of_Unique_Image_Features_by_Using_Camera_with_Controllable_Parameters)

8. Gonzalez, R.C., & Woods, R.E. (2008). Digital Image Processing. Upper Saddle River, NJ: Pearson Prentice Hall.
9. Fukui, K., & Yamaguchi, O. (2002). Face recognition using multi-viewpoint patterns for multi-camera. International Conference on Pattern Recognition (ICPR), 4, 527-530. URL: (<https://ieeexplore.ieee.org/document/1017623>)
10. Duda, R. O., & Hart, P. E. (1972). Use of the Hough Transformation to Detect Lines and Curves in Pictures. Communications of the ACM, 15(1), 11-15.
11. Ojala, T., Pietikainen, M., & Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. Pattern Recognition, 29(1), 51-59.
12. Aydin, T., & Akgul, Y.S. (2010). An occlusion insensitive adaptive focus measurement method. Optics Express, 18(13), 14212-14220. URL: (<https://opg.optica.org/abstract.cfm?uri=oe-18-13-14212>)
13. Zhang, H., Sun, Z., & Tan, T. (2010). Contact lens detection based on weighted LBP. 20th International Conference on Pattern Recognition (ICPR). URL:(<https://ieeexplore.ieee.org/abstract/document/5597756/>)
14. Zhang, S., Shen, Z., & Lin, Y. (2021). Removing Foreground Occlusions in Light Field using Micro-lens Dynamic Filter. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). URL:

(<https://scholar.archive.org/work/chw7j66inrffpo2h7fvq4o2qji/access/wayback/https://www.ijcai.org/proceedings/2021/0180.pdf>)

15. Zhou, Y., Liang, D., Chen, S., & Huang, S.J. (2023). Improving Lens Flare Removal with General-Purpose Pipeline and Multiple Light Sources Recovery. International Conference on Computer Vision (ICCV). URL: (http://openaccess.thecvf.com/content/ICCV2023/html/Zhou_Improving_Lens_Flare_Removal_with_General-Purpose_Pipeline_and_Multiple_Light_ICCV_2023_paper.html)

16. Dai, Y., Li, C., Zhou, S., & Feng, R. (2022). Flare7k: A phenomenological nighttime flare removal dataset. Advances in Neural Information Processing Systems (NeurIPS).

URL:(https://proceedings.neurips.cc/paper_files/paper/2022/hash/1909ac72220bf5016b6c93f08b66cf36-Abstract-Datasets_and_Benchmarks.html)

17. Vitoria, P., & Ballester, C. (2019). Automatic flare spot artifact detection and removal in photographs. Journal of Mathematical Imaging and Vision, 61(2), 146-158. URL: (<https://link.springer.com/article/10.1007/s10851-018-0859-0>)

ДОДАТОК А

Робоча програма для Розділу 2:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def display_image_with_variance(image_path):
    # Завантажити зображення
    image = cv2.imread(image_path)
    # Конвертувати зображення у відтінки сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Обчислити Лапласіан зображення
    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
    # Варіативність Лапласіана (дисперсія значень Лапласіана)
    variance = laplacian.var()
    if variance < 100:
        print(f"Зображення не сфокусоване: {variance:.2f}")
    else:
        print(f"Зображення сфокусоване: {variance:.2f}")
    # Візуалізація зображення та виведення варіативності
    plt.figure(figsize=(10, 5))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(f"Variance of Laplacian: {variance:.2f}")
    plt.axis('off') # Вимкнути осі
    plt.show()

# Приклад виклику функції
display_image_with_variance("/Users/maksimvasiliev/Desktop/22.png")
display_image_with_variance("/Users/maksimvasiliev/Desktop/Снимок экрана 2024-05-09 в 15.51.54.png")
```

ДОДАТОК Б

Робоча програма для Розділу 3:

```
import cv2
import numpy as np
from skimage import feature
import matplotlib.pyplot as plt

def load_image(image_path):
    """ Завантаження зображення і перетворення його в відтінки сірого. """
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image not found or path is incorrect")
    return image

def apply_lbp(image, radius=1, n_points=8):
    """ Застосування Local Binary Pattern до зображення. """
    lbp = feature.local_binary_pattern(image, n_points, radius, method="uniform")
    return lbp

def display_images(original, transformed, title_transformed="LBP Image"):
    """ Відображення оригінального і обробленого зображень. """
    plt.figure(figsize=(12, 6))
    plt.subplot(121)
    plt.imshow(original, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(122)
    plt.imshow(transformed, cmap='gray')
    plt.title(title_transformed)
    plt.axis('off')
    plt.show()

# Демонстраційні функції
image = load_image('/Users/maksimvasiliev/Desktop/Снимок экрана 2024-04-14 в 18.05.28.png')
lbp_image = apply_lbp(image)
display_images(image, lbp_image)
```

ДОДАТОК В

Робоча програма для Розділу 3:

```
import cv2
import numpy as np
from skimage import feature
import matplotlib.pyplot as plt

def load_image(image_path):
    """ Завантаження зображення і перетворення його в відтінки сірого. """
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image not found or path is incorrect")
    return image

def apply_lbp(image, radius=1, n_points=8):
    """ Застосування Local Binary Pattern до зображення. """
    lbp = feature.local_binary_pattern(image, n_points, radius, method="uniform")
    return lbp

def threshold_lbp(lbp, method='fixed', threshold=0.5):
    """ Бінаризація LBP-зображення на основі порога з попереднім перетворенням типу даних. """
    # Нормалізація LBP зображення до діапазону 0-255
    lbp_normalized = cv2.normalize(lbp, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX)
    lbp_uint8 = np.uint8(lbp_normalized) # Конвертація в 8-бітний тип
    if method == 'fixed':
        _, binary_image = cv2.threshold(lbp_uint8, threshold * 255, 255, cv2.THRESH_BINARY)
    elif method == 'otsu':
        _, binary_image = cv2.threshold(lbp_uint8, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return binary_image

def detect_contamination(binary_image):
    """ Визначення наявності забруднення на зображенні. """
```

```

    contamination_presence = np.any(binary_image > 0)
    contamination_area = np.sum(binary_image > 0) / binary_image.size
    return contamination_presence, contamination_area
def visualize_contamination(original, contamination_map):
    """ Візуалізація областей забруднення на оригінальному зображенні. """
    highlighted = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
    highlighted[contamination_map > 0] = [0, 0, 255] # Візуалізація забруднення червоним
    кольором
    return highlighted

# Оновлений приклад використання функцій для обробки зображення
image_path = '/Users/maksimvasiliev/Desktop/Снимок экрана 2024-04-14 в 18.05.28.png'
image = load_image(image_path)
lbp_image = apply_lbp(image)
binary_image = threshold_lbp(lbp_image, method='otsu')
contamination_present, contamination_area = detect_contamination(binary_image)
result_image = visualize_contamination(image, binary_image)
plt.imshow(result_image)
plt.title(f'Камера забруднена: {contamination_present}, Площа забрудненості:
{contamination_area:.2%}')
plt.show()

```

ДОДАТОК Г

Робоча програма для Розділу 3:

```
import cv2
import numpy as np
from skimage import feature
import matplotlib.pyplot as plt

def load_image(image_path):
    """ Завантаження зображення і перетворення його в відтінки сірого. """
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image not found or path is incorrect")
    return image

def apply_lbp(image, radius=1, n_points=8):
    """ Застосування Local Binary Pattern до зображення. """
    lbp = feature.local_binary_pattern(image, n_points, radius, method="uniform")
    return lbp

def threshold_lbp(lbp, method='fixed', threshold=0.5):
    """ Бінаризація LBP-зображення на основі порога з попереднім перетворенням типу даних. """
    # Нормалізація LBP зображення до діапазону 0-255
    lbp_normalized = cv2.normalize(lbp, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX)
    lbp_uint8 = np.uint8(lbp_normalized) # Конвертація в 8-бітний тип
    if method == 'fixed':
        _, binary_image = cv2.threshold(lbp_uint8, threshold * 255, 255, cv2.THRESH_BINARY)
    elif method == 'otsu':
        _, binary_image = cv2.threshold(lbp_uint8, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return binary_image
```

```

def detect_contamination(binary_image):
    """ Визначення наявності забруднення на зображенні. """
    contamination_presence = np.any(binary_image > 0)
    contamination_area = np.sum(binary_image > 0) / binary_image.size
    return contamination_presence, contamination_area

def visualize_contamination(original, contamination_map):
    """ Візуалізація областей забруднення на оригінальному зображенні. """
    highlighted = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
    highlighted[contamination_map > 0] = [0, 0, 255] # Візуалізація забруднення червоним
    кольором
    return highlighted

def detect_significant_contamination(binary_image, min_contamination_area=0.01):
    """ Визначення значущих областей забруднення з урахуванням мінімальної площі. """
    num_labels, labels, stats, _ = cv2.connectedComponentsWithStats(binary_image,
    connectivity=8)
    significant_areas = [stat[:4] for stat in stats if stat[-1] >= min_contamination_area *
    binary_image.size]
    return significant_areas

def preprocess_image(image, filter_size=5):
    """ Застосування фільтрації шуму до зображення. """
    return cv2.GaussianBlur(image, (filter_size, filter_size), 0)

def morphological_operations(binary_image, kernel_size=5):
    """ Застосування морфологічних операцій для поліпшення бінарного зображення. """
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    cleaned = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel) # Erosion
    followed by Dilation
    return cleaned

def calculate_contamination_area(binary_image):

```

```

""" Розрахунок відсотка площі забруднення від загальної площі зображення. """
area = np.sum(binary_image > 0) / binary_image.size
return area

try:
    # Завантаження та передобробка зображення
    image = load_image('/Users/maksimvasiliev/Desktop/Снимок екрана 2024-04-16 в
11.05.46.png')
    image_preprocessed = preprocess_image(image)
    # Застосування LBP
    lbp_image = apply_lbp(image_preprocessed)
    # Порогова обробка LBP-зображення з використанням методу Otsu
    binary_image = threshold_lbp(lbp_image, method='otsu')
    # Морфологічні операції
    binary_image_cleaned = morphological_operations(binary_image)
    # Розрахунок площі забруднення
    contamination_area = calculate_contamination_area(binary_image_cleaned)

    # Перевірка порогу забруднення
    contamination_threshold = 0.1 # 10% поріг
    contamination_present = contamination_area > contamination_threshold
    # Візуалізація забруднених областей
    result_image = visualize_contamination(image, binary_image_cleaned)
    # Виведення результатів
    plt.imshow(result_image)
    plt.title(f'Камера забруднена: {contamination_present}, Площа забрудненості:
{contamination_area:.2%}')
    plt.show()
except Exception as e:
    print(f'An error occurred: {e}')

```

ДОДАТОК Д

Робоча програма для Розділу 4:

```
import cv2
import numpy as np
def optimize_drop_detection(image_path):
    # Завантаження зображення
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    # Перетворення в градації сірого
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Підвищене розмиття для зменшення слабких контурів
    blurred = cv2.GaussianBlur(gray, (9, 9), 0)
    # Використання Hough Circle Transform для виявлення кіл
    # Підвищуємо param2 для зменшення помилкових позитивів
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1.2, minDist=50,
    param1=55, param2=60, minRadius=20,maxRadius=150)
    contamination_detected = False
    if circles is not None:
        circles = np.uint16(np.around(circles))
        contamination_detected = True
        for i in circles[0, :]:
            # Малюємо зовнішнє коло
            cv2.circle(img, (i[0], i[1]), i[2], (0, 255, 0), 2)
            # Малюємо центр кола
            cv2.circle(img, (i[0], i[1]), 2, (0, 0, 255), 3)
    if contamination_detected:
        print("На об'єктиві присутні краплі.")
    else:
        print("Об'єктив чистий.")
    # Показати зображення з відміченими краплями
    cv2.imshow('Optimized Drop Detection', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
optimize_drop_detection('/Users/maksimvasiliev/Desktop/каплі3.png')
```

ДОДАТОК Е

Робоча програма для Розділу 5:

```
import cv2
import numpy as np

def detect_occlusion(image_path):
    image = cv2.imread(image_path)
    if image is None:
        return "Image not found"
    # Конвертація в градації сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Використання порогової обробки для виявлення потенційно затулених областей
    _, thresholded = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY_INV)
    # Пошук контурів
    contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    occluded_area = sum(cv2.contourArea(contour) for contour in contours)
    total_area = image.shape[0] * image.shape[1]
    occlusion_image = cv2.drawContours(image.copy(), contours, -1, (0,255,0), 3)
    # Розрахунок відсотка затулення
    occlusion_percentage = (occluded_area / total_area) * 100
    if occlusion_percentage > 50:
        result = "Камера затулена"
    else:
        result = "Камера не затулена"
    return result, occlusion_image, occlusion_percentage

result, occlusion_image, occlusion_percentage =
detect_occlusion("/Users/maksimvasiliev/Desktop/Снимок экрана 2024-05-09 в 15.51.54.png")
print(result, occlusion_percentage)
cv2.imshow("Occlusion Detection", occlusion_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

ДОДАТОК Є

Робоча програма для Розділу 6:

```
import cv2
import numpy as np
def detect_lens_flare(image_path):
    # Завантаження зображення
    image = cv2.imread(image_path)
    if image is None:
        return "Image not found."
    # Конвертація в відтінки сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Порогова обробка для виділення дуже яскравих областей
    _, thresholded = cv2.threshold(gray, 235, 255, cv2.THRESH_BINARY)
    # Знаходження контурів
    contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # Площа яскравих областей та площа всього зображення
    flare_area = sum(cv2.contourArea(cnt) for cnt in contours)
    total_area = image.shape[0] * image.shape[1]
    flare_percentage = (flare_area / total_area) * 100
    # Візуалізація результатів
    flare_highlighted = image.copy()
    cv2.drawContours(flare_highlighted, contours, -1, (0, 255, 0), 3)
    cv2.imshow("Flare Highlighted", flare_highlighted)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    # Перевірка чи є засвітлення та вивід результатів
    flare_detected = print(f"Об'єktiv засвітлено, відсоток засвітлення: {flare_percentage}%") if
flare_percentage > 1.7 else print(f"Об'єktiv не засвітлено, відсоток
засвітлення: {flare_percentage}%")
    # Приклад виклику функції:
    detect_lens_flare("/Users/maksimvasiliev/Desktop/12.png")
```

ДОДАТОК Ж

Фінальний код програми для Розділу 7:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import feature

def detect_object(input_image):
    image = cv2.imread(input_image)
    plt.figure(figsize=(10, 5))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Вхідне зображення")
    plt.axis('off') # Вимкнути осі
    plt.show()
    print("Висновки після обробки зображення:")
    #####ДОДАТОК А#####

def display_image_with_variance(image_path):
    # Завантажити зображення
    image = cv2.imread(image_path)
    # Конвертувати зображення у відтінки сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Обчислити Лапласіан зображення
    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
    # Варіативність Лапласіана (дисперсія значень Лапласіана)
    variance = laplacian.var()
    if variance < 100:
        print(f"Камера не сфокусована: {variance:.2f}")
        plt.figure(figsize=(10, 5))
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title("Фокусування камери")
        plt.axis('off') # Вимкнути осі
        plt.show()
    else:
```

```

        print(f"Камера сфокусована: {variance:.2f}")
    # Візуалізація зображення та виведення варіативності
    display_image_with_variance(input_image)
    #####          ДОДАТОК Г          #####
def punkt3(input_image1):
    def load_image(image_path):
        """ Завантаження зображення і перетворення його в відтінки сірого. """
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            raise ValueError("Image not found or path is incorrect")
        return image
    def apply_lbp(image, radius=1, n_points=8):
        """ Застосування Local Binary Pattern до зображення. """
        lbp = feature.local_binary_pattern(image, n_points, radius, method="uniform")
        return lbp
    def threshold_lbp(lbp, method='fixed', threshold=0.5):
        """ Бінаризація LBP-зображення на основі порога з попереднім перетворенням
типу даних. """
        # Нормалізація LBP зображення до діапазону 0-255
        lbp_normalized = cv2.normalize(lbp, None, alpha=0, beta=255,
norm_type=cv2.NORM_MINMAX)
        lbp_uint8 = np.uint8(lbp_normalized) # Конвертація в 8-бітний тип
        if method == 'fixed':
            _, binary_image = cv2.threshold(lbp_uint8, threshold * 255, 255,
cv2.THRESH_BINARY)
        elif method == 'otsu':
            _, binary_image = cv2.threshold(lbp_uint8, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
        return binary_image
    def visualize_contamination(original, contamination_map):
        """ Візуалізація областей забруднення на оригінальному зображенні. """
        highlighted = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
        highlighted[contamination_map > 0] = [0, 0, 255] # Візуалізація забруднення
червоним кольором

```

```

        return highlighted
def preprocess_image(image, filter_size=5):
    """ Застосування фільтрації шуму до зображення. """
    return cv2.GaussianBlur(image, (filter_size, filter_size), 0)
def morphological_operations(binary_image, kernel_size=5):
    """ Застосування морфологічних операцій для поліпшення бінарного
зображення. """
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    cleaned = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel) #
Erosion followed by Dilation
    return cleaned
def calculate_contamination_area(binary_image):
    """ Розрахунок відсотка площі забруднення від загальної площі зображення.
"""

    area = np.sum(binary_image > 0) / binary_image.size
    return area
try:
    # Завантаження та передобробка зображення
    image = load_image(input_image1)
    image_preprocessed = preprocess_image(image)
    # Застосування LBP
    lbp_image = apply_lbp(image_preprocessed)
    # Порогова обробка LBP-зображення з використанням методу Otsu
    binary_image = threshold_lbp(lbp_image, method='otsu')
    # Морфологічні операції
    binary_image_cleaned = morphological_operations(binary_image)
    # Розрахунок площі забруднення
    contamination_area = calculate_contamination_area(binary_image_cleaned)
    # Перевірка порогу забруднення
    contamination_threshold = 0.1 # 10% попiр
    contamination_present = contamination_area > contamination_threshold
    # Візуалізація забруднених областей
    result_image = visualize_contamination(image, binary_image_cleaned)
    if contamination_area > 0.1:

```

```

        print(f'Камера забруднена: {contamination_present}, Площа забрудненості:
{contamination_area:.2%}')
        plt.imshow(result_image)
        plt.title(f'Камера забруднена: {contamination_present}, Площа
забрудненості: {contamination_area:.2%}')
        plt.show()
    else:
        print(f'Камера не забруднена, площа забрудненості:
{contamination_area:.2%}')
        # Виведення результатів
except Exception as e:
    print(f'An error occurred: {e}')
punkt3(input_image)
##### ДОДАТОК Д #####
def optimize_drop_detection(image_path):
    # Завантаження зображення
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    # Перетворення в градації сірого
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Підвищене розмиття для зменшення слабких контурів
    blurred = cv2.GaussianBlur(gray, (9, 9), 0)
    # Використання Hough Circle Transform для виявлення кіл
    # Підвищуємо param2 для зменшення помилкових позитивів
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1.3, minDist=100,
param1=55, param2=55, minRadius=50,
                                maxRadius=200)
    contamination_detected = False
    if circles is not None:
        circles = np.uint16(np.around(circles))
        # Намалювати круги на оригінальному зображенні
        contamination_detected = True
        for i in circles[0, :]:
            # Малюємо зовнішнє коло
            cv2.circle(img, (i[0], i[1]), i[2], (0, 255, 0), 2)

```

```

        # Малюємо центр кола
        cv2.circle(img, (i[0], i[1]), 2, (0, 0, 255), 3)
    if contamination_detected:
        print("На об'єктиві присутні краплі.")
        cv2.imshow("Water bubbls on camera", img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print("Краплі на об'єктиві відсутні.")

    # Показати зображення з відміченими краплями
optimize_drop_detection(input_image)
##### ДОДАТОК Е #####
def detect_occlusion(image_path):
    # Завантаження зображення
    image = cv2.imread(image_path)
    if image is None:
        return "Image not found"
    # Конвертація в градації сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Використання порогової обробки для виявлення потенційно затулених областей
    _, thresholded = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY_INV)
    # Пошук контурів
    contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # Визначення площі затулення
    occluded_area = sum(cv2.contourArea(contour) for contour in contours)
    total_area = image.shape[0] * image.shape[1]
    # Візуалізація затулених областей
    occlusion_image = cv2.drawContours(image.copy(), contours, -1, (0, 255, 0), 3)

    # Розрахунок відсотка затулення
    occlusion_percentage = (occluded_area / total_area) * 100
    # Висновок
    if occlusion_percentage > 50:

```

```

        print(f"Камера затулена, відсоток затулення: {occlusion_percentage}%")
        cv2.imshow("Occlusion camera", occlusion_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print(f"Камера не затулена, відсоток затулення: {occlusion_percentage}%")
    detect_occlusion(input_image)
##### ДОДАТОК Є #####
def detect_lens_flare(image_path):
    # Завантаження зображення
    image = cv2.imread(image_path)
    if image is None:
        return "Image not found."
    # Конвертація в відтінки сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Порогова обробка для виділення дуже яскравих областей
    _, thresholded = cv2.threshold(gray, 235, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # Площа яскравих областей та площа всього зображення
    flare_area = sum(cv2.contourArea(cnt) for cnt in contours)
    total_area = image.shape[0] * image.shape[1]
    flare_percentage = (flare_area / total_area) * 100
    flare_highlighted = image.copy()
    if flare_percentage > 4:
        print(f"Об'єktiv засвітлено, відсоток засвітлення: {flare_percentage} %")
        cv2.drawContours(flare_highlighted, contours, -1, (0, 255, 0), 3)
        cv2.imshow("Lens flare", flare_highlighted)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print(f"Об'єktiv не засвітлено, відсоток засвітлення: {flare_percentage} %")
    detect_lens_flare(input_image)
detect_object("/Users/maksimvasiliev/Desktop/каплі5.png")

```