Реферат

Листів \_31\_, илюстрацій \_6\_, джерел \_8\_, додатків \_2\_.

Основне завдання:

Дослідження та реалізація методів обробки відеоданих у системах управління з технічним зором за допомогою мови програмування Python та бібліотеки OpenCV.

Огляд проблеми обробки зображень в системах управління з технічним зором – розгляд методів та засобів отримання відеоданих, зокрема завантаження відео з файлу та захват відео з веб-камери.

Методи та засоби отримання відеоданих – розгляд методів та засобів отримання відеоданих, зокрема завантаження відео з файлу та захват відео з веб-камери.

Геометричні перетворення відео-зображень – розгляд геометричних перетвореннь відео-зображень, зокрема розглядається можливості бібліотеки OpenCV та метод bevel для скосу.

Кольорові перетворення відео-зображень – розгляд кольорових перетвореннь відео-зображень, зокрема розглядаються можливості бібліотеки OpenCV та метод colorspace\_change для зміни колірного простору.

Операції з відео-зображеннями – розгляд операцій з відео-зображеннями, зокрема можливості бібліотеки OpenCV для виконання операцій та метод canny\_edge\_detection для виділення меж з різними порогами.

Фільтрація відео-зображень – розгляд фільтрації відео-зображень, зокрема розглядаються можливості бібліотеки OpenCV та метод bilateral\_filtration для білатеральної фільтрації зображення.

Реалізація класу для обробки відеоданих – представлено реалізацію класу для обробки відеоданих – VideoProcessing.

КЮЧОВІ СЛОВА: технічний зір, геометричні перетворення, кольорові перетворення, виділення меж, фільтрація зображення.

# Зміст

[Перелік використаних позначень 4](#_Toc167947705)

[Вступ 5](#_Toc167947706)

[1 Огляд проблеми обробки зображень в системах управління з технічним зором 6](#_Toc167947707)

[2 Методи та засоби отримання відеоДАНИХ 9](#_Toc167947708)

[2.1 Завантаження відео з файлу 9](#_Toc167947709)

[2.2 Захват відео з веб-камери 9](#_Toc167947710)

[3 Геометричні перетворення відео-зображень 11](#_Toc167947711)

[3.1 Можливості бібліотеки OpenCV з геометричних перетворень 11](#_Toc167947712)

[3.2 Метод bevel для скосу 11](#_Toc167947713)

[3.3 Реалізація на Python й аналіз результатів 12](#_Toc167947714)

[4 Кольорові перетворення відео-зображень 13](#_Toc167947715)

[4.1 Можливості бібліотеки OpenCV з кольорових перетворень 13](#_Toc167947716)

[4.2 Метод colorspace\_change для зміни колірного простору 13](#_Toc167947717)

[4.3 Реалізація на Python й аналіз результатів 14](#_Toc167947718)

[5 ОПЕРАЦІЇ З ВІДЕО-ЗОБРАЖЕННЯМИ 15](#_Toc167947719)

[5.1 Можливості бібліотеки OpenCV для виконання операцій 15](#_Toc167947720)

[5.2 Метод canny\_edge\_detection для виділення меж з різними порогами 15](#_Toc167947721)

[5.3 Реалізація на Python й аналіз результатів 16](#_Toc167947722)

[6 ФІЛЬТРАЦІЯ ВІДЕО-ЗОБРАЖЕНЬ 17](#_Toc167947723)

[6.1 Можливості бібліотеки OpenCV з фільтрації 17](#_Toc167947724)

[6.2 Метод bilateral\_filtration для фільтрації зображення 17](#_Toc167947725)

[6.3 Реалізація на Python й аналіз результатів 18](#_Toc167947726)

[7 РЕАЛІЗАЦІЯ КЛАСУ ДЛЯ ОБРОБКИ ВІДЕОДАНИХ 19](#_Toc167947727)

[7.1 Поняття класу та його створення на Python 19](#_Toc167947728)

[7.2 Реалізація класу VideoProcessing 19](#_Toc167947729)

[Висновки 21](#_Toc167947730)

[Перелік використаних джерел 22](#_Toc167947731)

[Додаток А 23](#_Toc167947732)

[Додаток Б 30](#_Toc167947733)

# Перелік використаних позначень

ТЗ – технічний зір;

# Вступ

Системи технічного зору та обробки зображень є важливою галуззю сучасних технологій, що знаходить застосування в різних сферах життя. Ці системи дозволяють аналізувати, обробляти та інтерпретувати великі обсяги візуальної інформації шляхом застосування комп'ютерних алгоритмів та методів штучного інтелекту.

Системи технічного зору базуються на використанні різних типів сенсорів і камер для отримання візуальної інформації з навколишнього середовища. Ця інформація може бути оброблена за допомогою різних алгоритмів обробки зображень, які дозволяють виявляти об'єкти, визначати їх розташування, класифікувати образи та робити рішення на їх основі.

Тема роботи є актуальною в сучасному світі з ряду ключових причин. По-перше, з розвитком технологій в області штучного інтелекту, машинного навчання та глибокого навчання з'явилися нові можливості для аналізу та інтерпретації великих обсягів візуальних даних. Це дозволяє створювати більш точні та ефективні системи автоматизації, розпізнавання образів та забезпечення безпеки.

По-друге, зростання потреб у автоматизації процесів та розширення застосування штучного інтелекту приводить до зростання популярності систем технічного зору. Вони стають невід'ємною частиною систем автоматизації та інтелектуалізації виробничих процесів, транспортних систем, систем медичної діагностики та багатьох інших.

Робота досліджує можливості бібліотеки OpenCV для виконання операцій обробки зображень, таких як фільтрація, виявлення граней, зміна колірного простору тощо. Крім того, вона включає реалізацію класу для обробки відеоданих, який дозволяє вибирати різні режими обробки відео та надає можливість перегляду результатів обробки у реальному часі.

Отримана програма дозволяє користувачу зручно обробляти відеодані за допомогою різноманітних методів та алгоритмів, що вбудовані в клас VideoProcessing. Завдяки цій програмі користувач може легко виконувати такі операції, як зміна колірного простору, створення скосу зображення, виявлення граней за допомогою алгоритму Canny та білатеральне фільтрування.

# 1 Огляд проблеми обробки зображень в системах управління з технічним зором

Технічний зір (або машинний зір) (ТЗ)— це галузь комп'ютерних наук, яка займається розробкою технологій і систем, що дозволяють комп'ютерам сприймати, аналізувати та інтерпретувати візуальну інформацію з навколишнього середовища. Основна мета технічного зору полягає в тому, щоб надати комп'ютерам здатність "бачити" та розуміти світ, подібно до людей.

Відповідно до Асоціації автоматизованих зображень (AIA), ТЗ охоплює всі промислові та непромислові програми, у яких комбінація апаратного та програмного забезпечення забезпечує оперативне керівництво пристроями під час виконання їхніх функцій на основі захоплення та обробки зображень.

Там, де людський зір найкраще підходить для якісної інтерпретації складної, неструктурованої сцени, ТЗ перевершує кількісні вимірювання структурованої сцени завдяки своїй швидкості, точності та повторюваності [1].

В цілому значимість обробки зображень в системах управління полягає в її здатності забезпечити автоматизований контроль, нагляд та аналіз над різноманітними процесами. Обробка зображень дозволяє виявляти аномалії, розпізнавати об'єкт, вимірювати параметри об'єктів та середовища, а також забезпечує прийняття рішень у реальному часі.

В системах управління з ТЗ використовуються різноманітні технології для обробки зображень, які дозволяють досягти високої точності, швидкості та ефективності. Однією з ключових технологій є цифрова обробка зображень, яка дозволяє виділяти необхідну інформацію, покращувати якість зображень і видаляти шум. Вона може бути корисною для автоматичного розпізнавання об'єктів, вимірювання розмірів, виявлення аномалій та багато іншого.

Одним із найбільш поширених методів обробки є фільтрація зображень. Вона застосовується для згладжування, підвищення контрастності та видалення шуму. Наприклад, медіанний фільтр використовується для зменшення шуму, а гауссовий фільтр для згладжування.

Також активно використовуються алгоритми сегментації, які дозволяють розділяти зображення на окремі об'єкти або області. Одними з найпопулярніших методів сегментації є методи порогового значення, що виділяють об'єкти на основі їх яскравості.

Також не менш важливими є алгоритми відстеження об'єктів, які дозволяють відслідковувати рухомі об'єкти на послідовностях зображень, що є важливим для багатьох систем управління, таких як безпілотні транспортні засоби та робототехнічні системи.

Але, як і у будь-якої іншої технології, у ТЗ є проблеми. Основні проблеми, пов'язані з обробкою зображень, є складними і багатогранними, оскільки вони охоплюють різні аспекти технічних і теоретичних підходів.

Однією з головних проблем є якість зображень, що використовуються для аналізу. Низька якість зображень, викликана шумами, розмиттям або недостатнім освітленням, значно ускладнює точність і надійність результатів обробки.

Ще однією критичною проблемою є обробка зображень в реальному часі. Високі вимоги до швидкодії системи вимагають ефективних алгоритмів, які можуть обробляти великі обсяги даних у дуже короткий проміжок часу. Водночас, забезпечення такої швидкості обробки часто вимагає значних обчислювальних ресурсів, що може бути викликом для систем з обмеженими можливостями.

Проблема інтерпретації та аналізу результатів є ще одним важливим аспектом. Система повинна не тільки виявляти і розпізнавати об'єкти, але й правильно інтерпретувати їх значення в контексті конкретного завдання управління. Неточності або помилки в інтерпретації можуть призвести до неправильних рішень і дій.

Тому використання систем ТЗ не є остаточним та беззаперечним вирішенням усіх проблем та задач, які зараз виконуються людиною. Але є істотні передумови, які дозволяють вважати, що в недалекому майбутньому деякі з цих проблем будуть вирішені. Тому треба також звернути увагу на переваги систем ТЗ. Так Перрі Вест у своїй книзі виділив такі переваги ТЗ

Зменшення браку та переробки – раннє виявлення проблеми або, навіть краще, здатність розпізнати, коли проблема виникне до того, як вона виникне, є одним із способів, як машинне зір допомагає знизити витрати.

Підвищення продуктивності – швидкість систем зору дозволяє виробничим лініям працювати швидше, якщо людям не потрібно встигати. Потрібно менше робочої сили, а зміна виробничої лінії відбувається швидше та плавніше.

Краща безпека – машинний зір може виконувати завдання, які становлять ризики для безпеки працівників, і дозволяє розташувати працівників там, де ризик для їхньої безпеки є меншим.

Покращений контроль процесів – разом із тим, що воно робить, машинне бачення може реєструвати корисні виробничі дані, що дозволяє керівництву виробництва завчасно помічати тенденції, які інакше можуть призвести до проблем.

Підвищення якості – перевірка машинним зором є більш надійною, ніж перевірка людиною; він вільний від нудьги, втоми та відволікання. Машинний зір, будучи кількісним, є більш точним, ніж людське судження, яке є суб’єктивним і може змінюватися. Систему машинного бачення можна налаштувати на проведення більшої кількості перевірок, ніж це було б практично для інспектора-людини [2].

Ознайомившись з основними теоретичними положеннями проблеми обробки зображень у системах ТЗ, можна перейти до більш конкретного опису методів обробки зображень та до практичної частини роботи.

# 2 Методи та засоби отримання відеоДАНИХ

## 2.1 Завантаження відео з файлу

Завантаження відео з файлу є першим кроком у процесі обробки відеоданих. Це важливий етап, який включає в себе відкриття відеофайлу, зчитування кадрів і підготовку даних для подальшої обробки. У сучасних системах технічного зору відео може завантажуватись з різних джерел (локальні файли, потокове відео з веб-камер або інтернет-джерела). Ми будемо використовувати локальні файли та потокове відео з веб-камери у якості джерел.

Основними завданнями при завантаженні відео з файлу є перевірка наявності та доступності файлу, правильне зчитування кожного кадру і обробку помилок, які можуть виникати під час зчитування.

Для реалізації завантаження відео з файлу на Python часто використовують бібліотеку OpenCV, яка надає широкий набір інструментів для обробки зображень та відео. Тому ми нею і скористуємося.

Для роботи з відео був створений клас VideoProcessing. В ньому описані такі методи, які дозволяють працювати з відео-зображеннями:

\_\_init\_\_(self, filename) – конструктор класу, де визначається ім'я файлу та поточний режим обробки відео.

change\_input(self, new\_filename) – метод, що дозволяє змінити джерело відео.

Зі словника sources ми отримаємо необхідний нам режим введення. Якщо це буде video1 або video2, то на виході ми побачимо відео з цих файлів.

Перелічені вище методи можна побачити в лістингу коду в дод. А. Виведення відео на екран у вихідному вигляді та обробленому можна побачити на рисунках у дод. Б.

## 2.2 Захват відео з веб-камери

Захват відео з веб-камери є важливим аспектом у багатьох додатках комп'ютерного зору, це і системи технічного зору, і системи безпеки, і відеоконференції та інтерактивні мультимедійні додатки. Веб-камери дозволяють отримувати відеопотік у реальному часі, що забезпечує оперативність обробки і аналізу зображень.

Для захвату відео з веб-камери використовується той же клас, що і для захвату з файлу.

\_\_init\_\_(self, filename) – конструктор класу, де визначається ім'я файлу та поточний режим обробки відео.

change\_input(self, new\_filename) – метод, що дозволяє змінити джерело відео.

Зі словника sources ми отримаємо необхідний нам режим введення. Якщо це буде web, то на виході ми побачимо відео з веб-камери.

Перелічені вище методи можна побачити в лістингу коду в дод. А. Виведення відео на екран у вихідному вигляді та обробленому можна побачити на рисунках у дод. Б.

# 3 Геометричні перетворення відео-зображень

## 3.1 Можливості бібліотеки OpenCV з геометричних перетворень

Бібліотека OpenCV є потужним інструментом для обробки зображень та відео, що широко використовується в галузі комп'ютерного зору та машинного навчання. Вона надає різноманітні функції для геометричних перетворень зображень, які можуть бути корисними для корекції, обробки та аналізу зображень та відео.

Геометричні перетворення є ключовими операціями, що дозволяють змінювати форму, розмір, положення та орієнтацію зображень.

OpenCV надає можливість змінювати розмір зображення (зменшення або збільшення) та обрізати його до певних розмірів. Він також надає можливості виконання афінних та проективних перетворень, таких як зсув, масштабування, відображення та перспективна трансформація зображення.

Загальною перевагою використання OpenCV для геометричних перетворень є те, що вона надає широкий набір функцій і методів, які дозволяють легко і ефективно виконувати різноманітні операції з відео-зображеннями. Вони широко використовуються в таких областях як комп'ютерний зір, розпізнавання образів, відеоспостереження, медичне зображення тощо.

## 3.2 Метод bevel для скосу

Метод bevel здійснює афінне перетворення зображення, а саме зміну координат пікселів на зображенні. Це досягається шляхом застосування матриці афінного перетворення до координат пікселів вихідного зображення. Афінне перетворення може включати операції зсуву, повороту, масштабування та нахилу.

Параметри coef\_1 та coef\_2 у bevel(image, coef\_1=1.5, coef\_2=0.3) використовуються для визначення розміру та напрямку нахилу. Чим більше значення цих коефіцієнтів, тим більшим буде нахил.

У випадку методу bevel, матриця перетворення генерується за допомогою точок, що визначають початкову та кінцеву позиції для створення нахилу. Потім вона використовується для застосування нахилу до кожного пікселя зображення.

Після застосування методу bevel до зображення, пікселі будуть пересуватися залежно від заданих коефіцієнтів coef\_1 та coef\_2, створюючи враження нахилу або скосу.

## 3.3 Реалізація на Python й аналіз результатів

Даний метод є методом класу VideoProcessing. Вхідними даними є:

image: Вихідне зображення, до якого застосовується ефект нахилу.

coef\_1 – коефіцієнт, що визначає ступінь нахилу відносно горизонтальної вісі. Більше значення цього коефіцієнта призводить до більшого нахилу вправо.

coef\_2 – коефіцієнт, що визначає ступінь нахилу відносно вертикальної вісі. Більше значення цього коефіцієнта призводить до більшого нахилу вниз.

Принцип роботи:

1. Визначаються початкові та кінцеві точки для створення нахилу. Початкові точки встановлюються у верхньому лівому, верхньому правому та нижньому лівому куточках зображення.

2. Обчислюється кінцева позиція для кожної з цих точок, використовуючи введені коефіцієнти coef\_1 та coef\_2.

3. Генерується матриця перетворення за допомогою методу cv2.getAffineTransform, яка відображає початкові точки в кінцеві.

4. Застосовується афінне перетворення до зображення за допомогою методу cv2.warpAffine, що забезпечує нахил зображення згідно з заданими коефіцієнтами.

5. Згенероване нахилом зображення повертається як результат роботи методу.

В результаті роботи методу отримується зображення, до якого застосований нахил.

Даний метод можна побачити в лістингу коду в дод. А. Виведення відео на екран у вихідному вигляді та обробленому можна побачити на рис. Б.3. Як ми можемо бачити метод працює нормально.

# 4 Кольорові перетворення відео-зображень

## 4.1 Можливості бібліотеки OpenCV з кольорових перетворень

Бібліотека OpenCV надає широкі можливості для кольорових перетворень відео-зображень. Ці можливості дозволяють змінювати колірні простори зображення, виконувати кольорові фільтрації, змінювати насиченість, яскравість та інші параметри кольору.

Л. О. Краснов та О. В. Гавриленко в своєму навчальному посібнику так описують роботу бібліотеки з кольоровими перетвореннями:

«Бібліотека OpenCV надає широкі можливості для кольорових перетворень відео-зображень. Ці можливості дозволяють змінювати колірні простори зображення, виконувати кольорові фільтрації, змінювати насиченість, яскравість та інші параметри кольору.

Повнокольорове RGB-зображення (Red, Green, Blue) – це масив M×N×3, що складається з трьох матриць розміром M×N.

Ці матриці відповідають трьом колірним компонентам: червоній, зеленій і синій. Три монохромних зображення, що формують єдине RGB-зображення, називають червоною, зеленою і синьою компонентами зображення. [3]»

Тобто при деякому впливі на цю матрицю ми отримуємо різні результати. Бібліотека підтримує такі колірні моделі: RGB, HSV, YUV тощо.

## 4.2 Метод colorspace\_change для зміни колірного простору

Як написано на офіційному сайті OpenCV метод «перетворює вхідне зображення з одного колірного простору в інший [5].»

Принцип роботи методу полягає в застосуванні матриці перетворення кольорів до кожного пікселя зображення. Матриця перетворення визначається вибраним перетворенням між двома колірними просторами.

Наприклад, якщо ми хочемо перетворити зображення з колірного простору BGR у колірний простір HSV, ми можемо використати cv2.COLOR\_BGR2HSV як код перетворення. Виклик методу cv2.cvtColor() з цим кодом перетворення застосовує відповідну матрицю перетворення до кожного пікселя зображення.

Цей метод здатний працювати швидко і ефективно, оскільки внутрішні реалізації OpenCV оптимізовані для швидкого оброблення зображень у реальному часі.

## 4.3 Реалізація на Python й аналіз результатів

Метод colorspace\_change використовується для зміни колірного простору вхідного зображення на один із попередньо визначених колірних просторів. Принцип його роботи полягає у використанні функції cv2.cvtColor() з бібліотеки OpenCV для перетворення колірного простору зображення з одного типу в інший.

Вхідними даними є: image – вихідне зображення, яке необхідно перетворити; colorspace\_index – назва (або індекс) цільового колірного простору, у який потрібно перетворити зображення.

Передавши вхідне зображення image та цільовий колірний простір colorspace\_index, метод викликає функцію cv2.cvtColor() з параметрами image та colorspace\_index. Внутрішньо cv2.cvtColor() застосовує відповідну матрицю перетворення кольорів до кожного пікселя зображення. Ця матриця перетворення визначається вибраним перетворенням між двома колірними просторами.

Результатом роботи методу є зображення у новому колірному просторі, яке повертається як вихідна величина.

Даний метод можна побачити в лістингу коду в дод. А. Виведення відео на екран у вихідному вигляді та обробленому можна побачити на рис. Б.4. Як ми можемо бачити метод працює нормально.

# 5 ОПЕРАЦІЇ З ВІДЕО-ЗОБРАЖЕННЯМИ

## 5.1 Можливості бібліотеки OpenCV для виконання операцій

OpenCV містить реалізації різних алгоритмів для виявлення границь, таких як алгоритм Кенні, алгоритм Собеля, алгоритм Лапласа та інші. Ці алгоритми дозволяють ефективно виявляти границі об'єктів на зображеннях за різних умов освітлення та фонових шумів.

Алгоритми виділення границь є ключовим інструментом в обробці зображень, оскільки вони дозволяють виявити різкі зміни інтенсивності пікселів, які відповідають контурам об'єктів на зображенні.

Алгоритм Собеля використовує маски для обчислення градієнтів зображення в кожній точці. Він реагує на різкі зміни інтенсивності, що допомагає виявити контури об'єктів. Цей алгоритм добре працює з реальними зображеннями та шумами.

Алгоритм Лапласа використовує другу похідну зображення для виявлення змін інтенсивності. Він часто використовується для виявлення дуже різких змін інтенсивності, таких як краї об'єктів. Однак він чутливий до шуму.

Алгоритм Кенні - це узагальнення алгоритму Собеля, яке включає додаткові кроки для виявлення і видалення фальшивих позитивів (неправильно визначених границь) та об'єднання сусідніх пікселів в контури. Він зазвичай вважається найбільш точним серед алгоритмів границь, але він більш вимогливий до обробки [5].

## 5.2 Метод canny\_edge\_detection для виділення меж з різними порогами

Згідно з посібником В. О. Лавера та О. М. Левчука алгоритм складається із п'яти кроків:

1. Згладжування. Розмиття зображення для видалення шуму.

2. Пошук градієнтів. Границі відмічаються там, де градієнт зображення набуває максимального значення.

3. Приглушення не-максимумів. Тільки локальні максимум відмічаються як межі.

4. Подвійна порогова фільтрація. Потенціальні межі відмічаються порогами.

5. Трасування області неоднозначності. Границі визначаються шляхом подавлення усіх країв, не зв'язаних із сильними границями [6].

Ці кроки можна обґрунтувати згідно з описом методу на офіційному сайті:

«Оскільки виявлення країв чутливе до шуму на зображенні, першим кроком є ​​видалення шуму на зображенні за допомогою фільтра Гауса 5x5.

Потім згладжене зображення фільтрується за допомогою ядра Sobel як у горизонтальному, так і у вертикальному напрямку, щоб отримати першу похідну в горизонтальному напрямку і вертикальному напрямку.

Після отримання величини та напрямку градієнта виконується повне сканування зображення, щоб видалити небажані пікселі, які можуть не становити краю. Для цього для кожного пікселя перевіряється, чи є він локальним максимумом у своєму околиці в напрямку градієнта.

На цьому етапі вирішується, які ребра є справжніми ребрами, а які ні. Для цього нам потрібні два порогові значення minVal і maxVal. Будь-які грані з градієнтом інтенсивності, що перевищує maxVal, обов’язково є ребрами, а ті, що нижчі за minVal, точно не є ребрами, тому їх відкидають. [4]»

## 5.3 Реалізація на Python й аналіз результатів

Вхідні дані: image – вихідне зображення, яке необхідно обробити для виявлення граней. t\_lower – нижній поріг для гистерезисного порогового значення. Пікселі з градієнтом нижче цього значення будуть відкинуті. t\_upper – верхній поріг для гистерезисного порогового значення. Пікселі з градієнтом вище цього значення будуть прийняті як границі.

Зображення перетворюється у відтінки сірого за допомогою функції cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY). Це робиться тому, що алгоритм Кенні працює з одноканальними зображеннями, де інтенсивність пікселів представлена одним значенням.

Після проходження кожного етапу описаного вище отримаємо двійкове зображення, де границі об'єктів представлені білими пікселями (значення 255), а фон — чорними пікселями (значення 0). Це зображення містить всі виявлені границі об'єктів на вихідному зображенні.

Даний метод можна побачити в лістингу коду в дод. А. Виведення відео на екран у вихідному вигляді та обробленому можна побачити на рис. Б.5. Як ми можемо бачити метод працює нормально.

# 6 ФІЛЬТРАЦІЯ ВІДЕО-ЗОБРАЖЕНЬ

## 6.1 Можливості бібліотеки OpenCV з фільтрації

Бібліотека OpenCV надає потужний набір інструментів для фільтрації зображень, що є критично важливими для багатьох застосувань у комп'ютерному зорі та обробці зображень. Фільтрація дозволяє виконувати різні операції, такі як видалення шуму, згладжування, підвищення різкості та виявлення границь.

У посібнику ця тема розкрита достатньо широко:

«У практиці цифрового оброблення зображень широко використовується маскова фільтрація. Зазвичай як маска застосовується множина вагових коефіцієнтів заданих в усіх точках околу, які симетрично оточують по точну точку кадру.»

Рівномірна маска має всі вагові коефіцієнти однаковими, що дозволяє зберігати середню яскравість зображення. Цей процес називається віконною фільтрацією, де фільтр переміщається по всіх елементах зображення і в кожній точці усереднює значення яскравості сусідніх точок.

Фільтри низьких частот (НЧ-фільтри) використовуються для зменшення шуму, оскільки шум є високочастотним сигналом. Вони усереднюють значення яскравості, зменшуючи вплив шуму.

Фільтри високих частот (ВЧ-фільтри) використовуються для підвищення чіткості зображення, підкреслюючи перепади яскравості на межах об'єктів [7].

OpenCV надає різні інструменти для реалізації цих фільтрів, що дозволяє легко застосовувати їх до зображень для покращення якості та виділення важливої інформації.

## 6.2 Метод bilateral\_filtration для фільтрації зображення

Білатеральна фільтрація, використовується для зменшення шуму в зображенні, зберігаючи при цьому різкі межі об'єктів. Це досягається шляхом врахування як просторової близькості, так і різниці в яскравості між пікселями під час усереднення. Білатеральна фільтрація є нелінійним, неітеративним фільтром, який комбінує два типи згладжування: просторове та фотометричне. Вона має такі параметри:

Filter Size (d) – діаметр пікселів, які будуть враховані для фільтрації. Це визначає розмір вікна, в межах якого будуть враховуватися сусідні пікселі.

Sigma Color (σ\_color) – відповідає за величину, на яку впливає різниця в кольорі. Чим більша ця величина, тим більше значення пікселів будуть враховані, навіть якщо вони сильно відрізняються за кольором.

Sigma Space (σ\_space) – визначає, наскільки сильно впливає відстань між пікселями на процес згладжування. Більші значення призводять до більшого усереднення на більшій відстані.

Метод bilateral\_filtration застосовується до зображення image, повертаючи відфільтроване зображення, де шум зменшений, а межі залишаються чіткими.

## 6.3 Реалізація на Python й аналіз результатів

Вхідні дані: image – зображення, яке потребує обробки. Це може бути кольорове або монохромне зображення, представлене у вигляді матриці пікселів. filter\_size – діаметр пікселів, які будуть враховані для фільтрації. Визначає розмір області, в межах якої будуть враховуватися сусідні пікселі. sigmaValues – значення для сигм (σ), що використовуються для фотометричного та просторового згладжування. Він значення визначає, наскільки сильно впливає різниця в кольорі та відстань між пікселями на процес згладжування.

Спочатку зображення передається в метод як вхідний параметр. Метод використовує функцію cv2.bilateralFilter з переданими параметрами filter\_size і sigmaValues. В кінці відфільтроване зображення повертається як результат виконання методу. Зображення стає більш гладким завдяки усередненню пікселів в межах обраного вікна. При цьому чіткі межі об'єктів залишаються незмінними, що дозволяє зберігати важливі деталі та структуру зображення.

Даний метод можна побачити в лістингу коду в дод. А. Виведення відео на екран у вихідному вигляді та обробленому можна побачити на рис. Б.5. Як ми можемо бачити метод працює нормально.

# 7 РЕАЛІЗАЦІЯ КЛАСУ ДЛЯ ОБРОБКИ ВІДЕОДАНИХ

## 7.1 Поняття класу та його створення на Python

З сайту з документацією мови Python «Класи надають засоби об’єднання даних і функціональних можливостей. Створення нового класу створює новий тип об’єкта, що дозволяє створювати нові примірники цього типу. Кожен екземпляр класу може мати атрибути, приєднані до нього для підтримки його стану. Екземпляри класу також можуть мати методи (визначені його класом) для зміни свого стану.»

Для створення класу в Python використовується ключове слово class. Після нього слідує назва класу та двокрапка. Усередині класу визначаються атрибути та методи.

Конструктор – це спеціальний метод, який автоматично викликається при створенні нового об'єкта класу. В Python це метод \_\_init\_\_. Він використовується для ініціалізації атрибутів об'єкта.

Прикладними елементами класу є:

Ініціалізація (конструктор) – метод \_\_init\_\_ задає початковий стан об'єкта.

Методи – звичайні функції, які визначають, що об'єкт може робити.

Атрибути екземпляра – властивості, специфічні для конкретного об'єкта.

Атрибути класу – властивості, спільні для всіх об'єктів класу.

Ґрунтуючись на цих положеннях створимо клас, який буде задовільняти завданню цієї роботи.

## 7.2 Реалізація класу VideoProcessing

7.2.1. Задачі які повинен виконувати клас

Клас VideoProcessing призначений для виконання різноманітних операцій обробки відеоданих. Основними задачами класу є:

- Вибір режиму обробки відео (зміна колірного простору, геометричні перетворення, виявлення граней, фільтрація).

- Обробка кожного кадру відео відповідно до обраного режиму.

- Відображення оригінального та обробленого кадрів відео в режимі реального часу.

- Забезпечення можливості зупинки процесу обробки.

7.2.2. Опис атрибутів класу

\_\_filename (str) – назва файлу або шлях до джерела відео. Використовується для завантаження відео.

\_\_mode (int) – поточний режим обробки відео. Визначає, яка операція буде виконуватись над відео (0 - базовий, 1 - зміна колірного простору, 2 - геометричні перетворення, 3 - виявлення граней, 4 - фільтрація).

\_\_mode\_to\_func (dict) – словник, що містить методи обробки відео для кожного режиму. Ключ - номер режиму, значення - метод обробки.

\_\_mode\_to\_step (dict) – словник, що відображає режими на їх назви. Ключ – номер режиму, значення – назва режиму.

\_\_stop\_flag (bool) - прапорець для зупинки процесу обробки відео. Використовується для завершення роботи.

7.2.3. Опис методів класу

\_\_init\_\_ (self, filename) – конструктор класу. Ініціалізує атрибути класу.

change\_input(self, new\_filename) – змінює джерело відео.

set\_mode(self, mode\_key) – встановлює поточний режим обробки відео.

get\_current\_mode\_name(self) – повертає назву поточного режиму обробки відео.

colorspace\_change(image, colorspace\_index='RGB') – метод для зміни колірного простору зображення.

bevel(image, coef\_1=1.5, coef\_2=0.3) – метод для створення нахилу на зображенні.

canny\_edge\_detection(image, t\_lower=100, t\_upper=200) – метод для виявлення граней за допомогою алгоритму Canny.

bilateral\_filtration(image, filter\_size=9, sigmaValues=75) – метод для білатерального фільтрування зображення.

start\_processing(self) – починає обробку відео.

\_\_state\_check(self) – перевіряє стан клавіатури для зміни режиму або зупинки процесу.

\_\_modify\_frame(self, input\_frame) – модифікує кадр відео відповідно до поточного режиму.

Повний лістинг коду можна знайти у дод. А. Екранні форми виконання завдань класу можна знайти у дод. Б.

# Висновки

Аналізуючи отримані результати роботи програми можна зробити декілька висновків. Перш за все, програма виявилася досить ефективною та зручною у використанні, оскільки надає користувачу можливість швидко та зручно обробляти відео за допомогою різноманітних методів та алгоритмів.

Кожен з режимів обробки (зміна колірного простору, створення нахилу на зображенні, виявлення граней за допомогою алгоритму Canny та білатеральне фільтрування) працює коректно і надає очікуваний результат. Крім того, програма має простий інтерфейс, що дозволяє легко зрозуміти, як використовувати всі функціональні можливості.

У результаті аналізу можна підтвердити, що клас VideoProcessing відповідає вимогам ефективного та зручного інструменту для обробки відеоданих. Він демонструє високу функціональність, надійність та зручність у використанні, що робить його важливим компонентом для розвитку та вдосконалення систем технічного зору та обробки зображень.

# Перелік використаних джерел

1. INTRODUCTION TO MACHINE VISION A guide to automating process & quality improvements. . Cognex. URL: <https://www.assemblymag.com/ext/resources/White_Papers/Sep16/Introduction-to-Machine-Vision.pdf> (дата звернення: 14.05.2024).
2. West P. Fundamentals of Machine Vision. San Jose, California 95124 U.S.A : AUTOMATED VISION SYSTEMS, INC., 2021 – 34 с. URL: <https://www.autovis.com/images/pdf/resources/fundamentals-of-machine-vision.pdf> (дата звернення: 18.05.2024).
3. Краснов Л. О., Гавриленко О. В. Об’єктно-орієнтоване проектування систем керування [Текст] : навч. посіб. / Л. О. Краснов, О. В. Гавриленко – Харків : Нац. аеро- косм. ун-т ім. М. Є. Жук. «Харків. авіац. ін-т», 2020 – 184 с. URL: <https://drive.google.com/drive/folders/1x4LfQbIycNv2k6YfuLOwpTwQanW4rt8z> (дата звернення: 21.05.2024).
4. Електронний ресурс: <http://docs.opencv.org> – Документація по бібліотеці OpenCV.
5. Празднікова М. О. Метод розпізнавання меж об'єктів за допомогою модифікованого алгоритму Канні : дис. … д-ра філософії в галузі техн. наук : 123. Київ, 2021 – 93 с. URL: <https://ela.kpi.ua/server/api/core/bitstreams/4cd90e83-5244-40d9-8c69-a45007856ebe/content> (дата звернення: 24.05.2024).
6. Лавер В. О., Левчук О. М. Обробка зображень: навч.-метод. посіб. / В. О. Лавер, О. М. Левчук. – Ужгород : ПП «АУТДОР - ШАРК», 2021 – 51 с. URL: <https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/35667/1/Обробка%20зображень%202021.pdf> (дата звернення: 20.05.2024).
7. Основи побудови сучасних мобільних систем технічного зору [Текст]: навч. посіб. (часть 2). / Л. О. Краснов, К. Ю. Дергачов, С. В. Багінський, Є. В. Пявка – Харків: Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харьк. авіац. ін-т», 2018. – 92 с.
8. ­­­­­­­­­­­­­­­­­­ Електронний ресурс: <https://www.python.org/> – Документація по мові програмування Python.

# Додаток А

Лістинг коду

Лістинг коду для зчитування відео з файлу:

# підключення необхідних бібліотек

import cv2

sources = {'video1': "video1.mp4", 'video2': "video2.mp4", "web": 0} # Словник із джерелами відео

def main():

cap = cv2.VideoCapture(sources.get('video1')) # Відкриття відеофайлу для читання

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read() # Зчитування кадру з відеопотоку

# При виникненні помилці запису

if not ret:

print("Frame recording error!") # Виведення повідомлення про помилку запису

break

cv2.namedWindow('frame', cv2.WINDOW\_NORMAL) # Створення та налаштування вікна відображення

cv2.resizeWindow('frame', 800, 700)

# Відображення результату

cv2.imshow('frame', frame) # Відображення кадру

if cv2.waitKey(25) == ord('q'): # Очікування натискання клавіші 'q' для виходу

break

# Завершуємо запис у кінці роботи

cap.release() # Закриття відеопотоку

cv2.destroyAllWindows() # Закриття всіх вікон OpenCV

# при запуску як головного файлу

if \_\_name\_\_ == '\_\_main\_\_':

main()

Лістинг коду для геометричногоперетворення відео з файлу:

# підключення необхідних бібліотек

import cv2

import numpy as np

sources = {'video1': "video1.mp4", 'video2': "video2.mp4", "web": 0} # Словник із джерелами відео

# функція скісу зображення

def bevel(image, coef\_1=1.0, coef\_2=0.0):

rows, cols = image.shape[:2]

# Визначаємо початкові і кінцеві точки для створення скосу

src\_points = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])

dst\_points = np.float32([[0, 0], [int(coef\_1 \* (cols - 1)), 0],

[int(coef\_2 \* (cols - 1)), rows - 1]])

affine\_matrix = cv2.getAffineTransform(src\_points, dst\_points) # Отримуємо матрицю перетворення

img\_output = cv2.warpAffine(image, affine\_matrix, (cols, rows)) # Застосовуємо скіс до зображення

return img\_output

def main():

cap = cv2.VideoCapture(sources.get('video1')) # Відкриття відеофайлу для читання

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read() # Зчитування кадру з відеопотоку

# При виникненні помилці запису

if not ret:

print("Frame recording error!") # Виведення повідомлення про помилку запису

break

# Геометричні перетворення зображення (фрейму)

frame\_rotate = rotate\_image(frame, 30)

frame\_transfer = parallel\_transfer(frame, 100)

frame\_bevel = bevel(frame, 0.6, 0.4)

frame\_mirror = mirror(frame)

# Створення та налаштування вікон відображення

cv2.namedWindow('frame', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame', 400, 350)

cv2.namedWindow('frame\_bevel', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame\_bevel', 400, 350)

# Відображення результату

cv2.imshow('frame', frame)

cv2.imshow('frame\_bevel', frame\_bevel)

if cv2.waitKey(25) == ord('q'):

break

# Завершуємо запис у кінці роботи

cap.release() # Закриття відеопотоку

cv2.destroyAllWindows() # Закриття всіх вікон OpenCV

# при запуску як головного файлу

if \_\_name\_\_ == '\_\_main\_\_':

main()

Лістинг коду для зміни колірного простору зображення відео з файлу:

# підключення необхідних бібліотек

import cv2

sources = {'video1': "video1.mp4", 'video2': "video2.mp4", "web": 0} # Словник із джерелами

colorspaces = {'Gray': cv2.COLOR\_BGR2GRAY, 'XYZ': cv2.COLOR\_BGR2XYZ, 'LAB': cv2.COLOR\_BGR2LAB,

'YUV': cv2.COLOR\_BGR2YUV, 'HSV': cv2.COLOR\_BGR2HSV} # Визначення колірних просторів для зміни

# Функція для зміни колірного простору зображення

def colorspace\_change(input\_frame, colorspace\_index):

return cv2.cvtColor(input\_frame, colorspaces.get(colorspace\_index))

def main():

cap = cv2.VideoCapture(sources.get('video1')) # Ініціалізація об'єкту для відеозахоплення з веб-камери

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Frame recording error!")

break

# Зміна колірного простору зображення (фрейму)

frame\_RGB = colorspace\_change(frame, 'RGB')

# Створення та налаштування вікон відображення

cv2.namedWindow('frame', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame', 400, 350)

cv2.namedWindow('frame RGB', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame RGB', 400, 350)

# Відображення результату

cv2.imshow('frame', frame)

cv2.imshow('frame RGB', frame\_RGB)

if cv2.waitKey(25) == ord('q'): # Очікування натискання клавіші 'q' для виходу

break

# Завершуємо запис у кінці роботи

cap.release() # Закриття відеопотоку

cv2.destroyAllWindows() # Закриття всіх вікон OpenCV

# при запуску як головного файлу

if \_\_name\_\_ == '\_\_main\_\_':

main()

Лістинг коду для виділення меж canny відео з файлу:

# підключення необхідних бібліотек

import cv2

import numpy as np

sources = {'video1': "video1.mp4", 'video2': "video2.mp4", "web": 0} # Словник із джерелами відео

# функція виділення меж canny

def canny\_edge\_detection(image, t\_lower=100, t\_upper=200):

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Перетворення зображення в відтінки сірого

edge = cv2.Canny(gray\_image, t\_lower, t\_upper) # Використання алгоритму Canny для виявлення граней

return edge

def main():

cap = cv2.VideoCapture(sources.get('video1')) # Відкриття відеофайлу для читання

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read() # Зчитування кадру з відеопотоку

# При виникненні помилці запису

if not ret:

print("Frame recording error!") # Виведення повідомлення про помилку запису

break

# Виконання операції за варіантом

frame\_changed = canny\_edge\_detection(frame, t\_lower=100, t\_upper=200)

# Створення та налаштування вікон відображення

cv2.namedWindow('frame', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame', 400, 350)

cv2.namedWindow('frame\_changed', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame\_changed', 400, 350)

# Відображення результату

cv2.imshow('frame', frame)

cv2.imshow('frame\_changed', frame\_changed)

if cv2.waitKey(25) == ord('q'): # Перевірка натискання клавіші 'q' для виходу з циклу

break

# Завершуємо запис у кінці роботи

cap.release()

cv2.destroyAllWindows()

# при запуску як головного файлу

if \_\_name\_\_ == '\_\_main\_\_':

main()

Лістинг коду для зчитування відео з файлу:

# підключення необхідних бібліотек

import cv2

import numpy as np

sources = {'video1': "video1.mp4", 'video2': "video2.mp4", "web": 0} # Словник із джерелами відео

# Білатеральний фільтр

def bilateral\_filtration(image, filter\_size=9, sigmaValues=75):

return cv2.bilateralFilter(image, filter\_size, sigmaValues, sigmaValues)

def main():

cap = cv2.VideoCapture(sources.get('web')) # Відкриття відеофайлу для читання

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Frame recording error!") # Виведення повідомлення про помилку запису

break

# Виконання операції за варіантом

frame\_changed1 = bilateral\_filtration(frame, filter\_size=9, sigmaValues=75)

frame\_changed2 = bilateral\_filtration(frame, filter\_size=10, sigmaValues=150)

frame\_changed3 = bilateral\_filtration(frame, filter\_size=2, sigmaValues=1)

# Створення та налаштування вікон відображення

cv2.namedWindow('frame', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame', 400, 350)

cv2.namedWindow('frame\_changed1', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame\_changed1', 400, 350)

cv2.namedWindow('frame\_changed2', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame\_changed2', 400, 350)

cv2.namedWindow('frame\_changed3', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame\_changed3', 400, 350)

# Відображення результату

cv2.imshow('frame', frame)

cv2.imshow('frame\_changed1', frame\_changed1)

cv2.imshow('frame\_changed2', frame\_changed2)

cv2.imshow('frame\_changed3', frame\_changed3)

if cv2.waitKey(25) == ord('q'): # Перевірка натискання клавіші 'q' для виходу з циклу

break

# Завершуємо запис у кінці роботи

cap.release()

cv2.destroyAllWindows()

# при запуску як головного файлу

if \_\_name\_\_ == '\_\_main\_\_':

main()

Повний лістинг коду:

import cv2

import numpy as np

sources = {'video1': "video1.mp4", 'video2': "video2.mp4", "web": 0}

colorspaces = {'Gray': cv2.COLOR\_BGR2GRAY, 'XYZ': cv2.COLOR\_BGR2XYZ, 'LAB': cv2.COLOR\_BGR2LAB,

'YUV': cv2.COLOR\_BGR2YUV, 'HSV': cv2.COLOR\_BGR2HSV, 'RGB': cv2.COLOR\_BGR2RGB} # Визначення колірних просторів для зміни

class VideoProcessing:

"""Video processing class"""

# конструктор

def \_\_init\_\_(self, filename):

self.\_\_filename = filename

self.\_\_mode = 0

# Словник, що містить методи обробки відео для кожного режиму

self.\_\_mode\_to\_func = {1: VideoProcessing.colorspace\_change, 2: VideoProcessing.bevel,

3: VideoProcessing.canny\_edge\_detection, 4: VideoProcessing.bilateral\_filtration}

# Словник, що відображає режими на їх назви

self.\_\_mode\_to\_step = {0: "Basic", 1: "Colorspace", 2: "Geometric", 3: "Operation", 4: "Filtration"}

self.\_\_stop\_flag = False

# Зміна джерела відео

def change\_input(self, new\_filename):

self.\_\_filename = new\_filename

# Встановлення поточного режиму обробки відео

def set\_mode(self, mode\_key):

self.\_\_mode = mode\_key

# Отримання назви поточного режиму

def get\_current\_mode\_name(self):

return self.\_\_mode\_to\_step.get(self.\_\_mode)

# Метод для зміни колірного простору

@staticmethod

def colorspace\_change(image, colorspace\_index='RGB'):

return cv2.cvtColor(image, colorspaces.get(colorspace\_index))

# Метод для створення нахилу на зображенні

@staticmethod

def bevel(image, coef\_1=1.5, coef\_2=0.3):

rows, cols = image.shape[:2]

# Визначаємо початкові і кінцеві точки для створення скосу

src\_points = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])

dst\_points = np.float32([[0, 0], [int(coef\_1 \* (cols - 1)), 0],

[int(coef\_2 \* (cols - 1)), rows - 1]])

affine\_matrix = cv2.getAffineTransform(src\_points, dst\_points) # Отримуємо матрицю перетворення

img\_output = cv2.warpAffine(image, affine\_matrix, (cols, rows)) # Застосовуємо скіс до зображення

return img\_output

# Метод для виявлення граней за допомогою алгоритму Canny

@staticmethod

def canny\_edge\_detection(image, t\_lower=100, t\_upper=200):

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Перетворення зображення в відтінки сірого

edge = cv2.Canny(gray\_image, t\_lower, t\_upper) # Використання алгоритму Canny для виявлення граней

return edge

# Метод для білатерального фільтрування зображення

@staticmethod

def bilateral\_filtration(image, filter\_size=9, sigmaValues=75):

return cv2.bilateralFilter(image, filter\_size, sigmaValues, sigmaValues)

# методи класу

def start\_processing(self):

cap = cv2.VideoCapture(self.\_\_filename)

self.\_\_stop\_flag = False

# Перевірка готовності веб-камери

while cap.isOpened() and not self.\_\_stop\_flag:

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Frame recording error!")

break

frame\_changed = self.\_\_modify\_frame(frame)

# Відображення результату

cv2.namedWindow('frame', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame', 400, 350)

cv2.namedWindow('frame\_changed', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('frame\_changed', 400, 350)

cv2.imshow('frame', frame)

cv2.imshow('frame\_changed', frame\_changed)

self.\_\_state\_check()

# Завершуємо запис у кінці роботи

cap.release()

cv2.destroyAllWindows()

# Перевірка стану клавіатури

def \_\_state\_check(self):

key\_code = cv2.waitKey(25)

if key\_code == ord('0'):

self.set\_mode(0)

elif key\_code == ord('1'):

self.set\_mode(1)

elif key\_code == ord('2'):

self.set\_mode(2)

elif key\_code == ord('3'):

self.set\_mode(3)

elif key\_code == ord('4'):

self.set\_mode(4)

elif key\_code == ord('q'):

self.\_\_stop\_flag = True

print(self.get\_current\_mode\_name())

# Модифікація кадру відео залежно від поточного режиму

def \_\_modify\_frame(self, input\_frame):

if self.\_\_mode == 0:

return input\_frame

elif self.\_\_mode in self.\_\_mode\_to\_func.keys():

return self.\_\_mode\_to\_func.get(self.\_\_mode)(input\_frame)

def main():

# Створення екземпляру класу та обробка відео

video\_processing = VideoProcessing(sources.get('video1'))

video\_processing.start\_processing()

# Зміна режиму та джерела відео

video\_processing.set\_mode(3)

video\_processing.change\_input(sources.get('video2'))

video\_processing.start\_processing()

if \_\_name\_\_ == '\_\_main\_\_':

main()

# Додаток Б

Екранні форми виконання

Изображение выглядит как ваза, стекло

Автоматически созданное описание

Рисунок Б.1 – Зчитування відео з файлу

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, компьютер

Автоматически созданное описание

Рисунок Б.2 – Зчитування відео з веб-камери

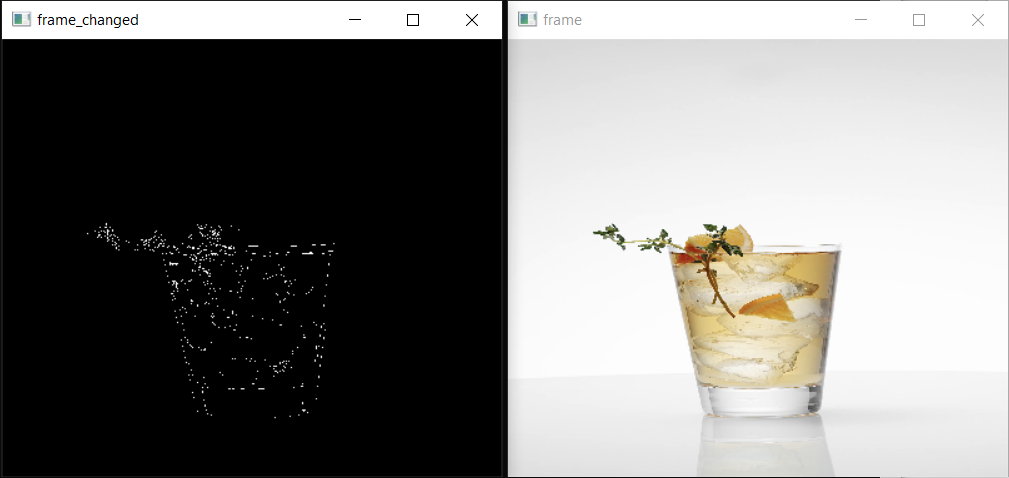
Изображение выглядит как кружка, стекло, еда

Автоматически созданное описание со средним доверительным уровнем

Рисунок Б.3 – Геометричне перетворення відео (скіс)

Изображение выглядит как стекло, безалкогольный напиток, пить, Сосуды для напитков

Автоматически созданное описаниеРисунок Б.4 – Кольорове перетворення відео (RGB)

Рисунок Б.5 – Операції з відео-зображенням (canny)

Изображение выглядит как стекло, ваза, контейнер

Автоматически созданное описаниеРисунок Б.6 – Відео з білатеральним фільтром