Реферат

Листів 35, ілюстрацій 7, джерел 7, додатків 2.

Ця курсова робота зосереджується на вивченні та оцінюванні можливостей обробки відео даних за допомогою фреймворку OpenCV в мові програмування Python. Робота містить кілька розділів, у яких розглядаються ключові техніки обробки відео, зокрема завантаження відео з файла або захоплення з відеокамери, кольорові перетворення (HSV колірний простір), геометричні перетворення, операції на відеозображенні (наприклад, виявлення країв за допомогою Canny з різними пороговими значеннями), фільтрація шумів (застосування фільтра зсуву з різними параметрами) і реалізація класу для обробки відеоданих, отриманих з файла або відеокамери. Кожен розділ включає в себе опис методології, її імплементацію у Python та аналіз результатів її застосування.

КЛЮЧОВІ СЛОВА: БІБЛІОТЕКА OPENCV, ОБРОБКА ВІДЕОДАНИХ, ГЕОМЕТРИЧНІ ТА КОЛІРОВІ ПЕРЕТВОРЕННЯ, ВИДІЛЕННЯ КРАЇВ, ФІЛЬТРУВАННЯ ШУМІВ, ТЕХНІЧНИЙ ЗІР, РОБОТА З ВІДЕОКАМЕРОЮ.

# ЗМІСТ

[ЗМІСТ 2](#_Toc166789865)

[ПЕРЕЛІК ВИКОРИСТАНИХ ПОЗНАЧЕНЬ 4](#_Toc166789866)

[ВСТУП 5](#_Toc166789867)

[1 ОГЛЯД ПРОБЛЕМИ ОБРОБКИ ЗОБРАЖЕНЬ В СИСТЕМАХ УПРАВЛІННЯ З ТЕХНІЧНИМ ЗОРОМ 7](#_Toc166789868)

[2 МЕТОДИ ТА ЗАСОБИ ОТРИМАННЯ ВІДЕОДАНИХ 9](#_Toc166789869)

[2.1 Завантаження відео з файлу 9](#_Toc166789870)

[2.2 Захват відео з веб-камери 10](#_Toc166789871)

[3 КОЛІРНІ ПЕРЕТВОРЕННЯ ВІДЕО-ЗОБРАЖЕНЬ 11](#_Toc166789872)

[3.1 Можливості бібліотеки OpenCV з колірних перетворень 11](#_Toc166789873)

[3.2 Метод зміни колірного перетворення для колірного простору HSV 11](#_Toc166789874)

[3.3 Реалізація на Python й аналіз результатів 12](#_Toc166789875)

[4 ГЕОМЕТРИЧНІ ПЕРЕТВОРЕННЯ ВІДЕО-ЗОБРАЖЕНЬ 13](#_Toc166789876)

[4.1 Можливості бібліотеки OpenCV з геометричних перетворень 13](#_Toc166789877)

[4.2 Метод геометричного перетворення для перенесення. 13](#_Toc166789878)

[4.3 Реалізація на Python й аналіз результатів 14](#_Toc166789879)

[5 ОПЕРАЦІЇ З ВІДЕО-ЗОБРАЖЕННЯМИ 15](#_Toc166789880)

[5.1 Можливості бібліотеки OpenCV для виконання операцій з відеозображеннями 15](#_Toc166789881)

[5.2 Метод операції з відео-зображеннями для виділення меж з різними порогами canny. 15](#_Toc166789882)

[5.3 Реалізація на Python й аналіз результатів 16](#_Toc166789883)

[6 ФІЛЬТРАЦІЯ ВІДЕО-ЗОБРАЖЕНЬ 18](#_Toc166789884)

[6.1 Можливості бібліотеки OpenCV з фільтрації 18](#_Toc166789885)

[6.2 Метод обробки відео-зображень для фільтрації з ефектом зсуву з різними параметрами 18](#_Toc166789886)

[6.3 Реалізація на Python й аналіз результатів 19](#_Toc166789887)

[7 РЕАЛІЗАЦІЯ КЛАСУ ДЛЯ ОБРОБКИ ВІДЕОДАНИХ 20](#_Toc166789888)

[7.1 Поняття класу та його створення на Python 20](#_Toc166789889)

[7.2 Реалізація класу для обробки відеоданих VideoProcessor 20](#_Toc166789890)

[Висновки 22](#_Toc166789891)

[Перелік використаних джерел 23](#_Toc166789892)

[Додаток А 24](#_Toc166789893)

[Додаток Б 36](#_Toc166789894)

ПЕРЕЛІК ВИКОРИСТАНИХ ПОЗНАЧЕНЬ

ТЗ – технічний зір;

ПЗ – програмне забезпечення;

OpenCV - Open Computer Vision Library

ООП – об’єктно орієнтоване програмування

RGB – Red, Green, Blue

HSV – Hue, Saturation, Value

# ВСТУП

Системи технічного зору використовуються в сучасному світі в різних галузях від виробничої промисловості до медицини. Вони дозволяють відцифровувати зображення, виділяти об'єкти, аналізувати їх форму та властивості, а також розпізнавати окремі об'єкти на зображенні. Обробка зображень є важливим аспектом технічного зору, який дозволяє перетворювати зображення, вилучати шум, визначати краї та виконувати інші дії.

Системи технічного зору впроваджуються в різні сфери людської діяльності, починаючи від автоматизації виробничих процесів і закінчуючи біомедичними дослідженнями. Вони використовуються для автоматичного виявлення, вимірювань, перевірки, розпізнавання образів та багато іншого. завдань.

Фундаментальною частиною будь-якої системи технічного зору є обробка зображень. Це процес вдосконалення зображень таким чином, щоб вони були придатні для подальшого аналізу. Це може включати в себе зменшення шуму, посилення деталей або виділення важливих структур на зображенні.

Можливості обробки зображень в системах комп'ютерного зору розширюються завдяки використанню сучасних алгоритмів та методів. Наприклад, бібліотека OpenCV (Open Source Computer Vision) - це один з найбільш популярних інструментів для обробки зображень та відео. Вона містить більше 2500 вдосконалених алгоритмів обробки зображень та машинного навчання, які можуть допомогти в різних сценаріях обробки зображень - від виявлення та розпізнавання облич, до створення панорамних зображень або відстеження рухів.

Використання бібліотеки OpenCV в рамках такого потужного і гнучкого інструменту, як Python, створює безмежні можливості для розробки, впровадження та дослідження систем комп'ютерного зору.

В цій курсовій роботі будуть розглянуті ключові аспекти роботи з технічним зором, зокрема, застосування OpenCV для обробки відео. Дослідженно виконання кольорових та геометричних перетворень, визначення країв, фільтрації шумів і багато інших. Надамо детальний опис кожного методу, приведемо його реалізацію на Python, а також проаналізуємо результати їхнього використання. Ця робота служить хорошим посібником для тих, хто хоче ускладнити свої навички роботи з обробкою відеоданих та системами технічного зору.

# 1 ОГЛЯД ПРОБЛЕМИ ОБРОБКИ ЗОБРАЖЕНЬ В СИСТЕМАХ УПРАВЛІННЯ З ТЕХНІЧНИМ ЗОРОМ

Системи технічного зору стали невід'ємною частиною сучасних процесів управління в різних областях, зокрема в промисловості, медицині, безпеці та багатьох інших. Однак, незалежно від задачі, якій служить система технічного зору, основою її функціонування є обробка зображень. Вони використовуються для автоматичного контролю якості, управління роботами, автоматизованого розпізнавання об'єктів, біометричної ідентифікації та багато іншого. Технічний зір надає можливість комп'ютерам сприймати та інтерпретувати візуальні дані, послаблюючи людську участь та можливість появи помилок. Він створює нові можливості для підвищення продуктивності та ефективності в різних секторах, працюючи безперервно та здатний обробляти великі обсяги даних. [1]

Завдяки розвитку і впровадженню складних алгоритмів, ми можемо обробляти зображення на такому рівні, який, можливо, був недосяжним лише декілька років тому. Завдяки розвитку і впровадженню складних алгоритмів, можливості обробки зображень досягли високого рівня. Нині дослідники та інженери мають доступ до широкого спектра технологій та інструментарію, включаючи конволюційні нейронні мережі, автоенкодери, GANs та інші методи глибокого навчання. Ці інструменти можуть моделювати складні взаємозв'язки в даних і генерувати деталізовану інформацію на основі вхідних зображень. Втім, незважаючи на значний прогрес в цій області, деякі проблеми з обробкою зображень в системах технічного зору досі залишаються актуальними. До них входять шум на зображеннях, складність деяких процесів обробки зображень та виклики у налаштуванні параметрів алгоритмів обробки зображень[1]. Однак, незважаючи на значний прогрес в цій області, деякі проблеми з обробкою зображень в системах технічного зору досі залишаються актуальними.

Однією з незаперечних проблем є шум, який може знизити якість вихідного зображення та призвести до неточностей під час його розпізнавання. Такий шум може виникати в результаті різних факторів, включаючи помилки обладнання, електромагнітні перешкоди, проблеми з освітленням та інше. Обробка шуму - це не проста задача, особливо при обробці зображень в режимі реального часу, де швидкість обробки відіграє важливу роль. Це вимагає ретельного аналізу та застосування методів фільтрації та згладжування для мінімізації впливу шуму. Використання адаптивних фільтрів, які можуть змінюватися відповідно до характеристик шуму та зображення, може бути одним з можливих рішень для цього виклику. [2]. Видалення шуму - це не проста задача, особливо в умовах реального часу, де швидкість обробки відіграє важливу роль.

Деякі задачі обробки зображень, такі як виявлення крайових областей та вирівнювання гістограм, вимагають істотних обчислювальних зусиль. Виявлення крайових областей є ключовим етапом в розпізнаванні об'єктів і структур на зображеннях, але це може бути викликом у випадках низького контрасту або шуму. Вирівнювання гістограми, яке допомагає підвищити контраст зображення, може бути ресурсозатратними, особливо для великих зображень. Ці фактори можуть знизити загальну продуктивність системи управління, тому вони вимагають оптимізації та вдосконалення. Важливо використовувати ефективні алгоритми та методи, які забезпечують потрібну точність, не втрачаючи при цьому в обчислювальній потужності [2]. Ці процеси вимагають істотних обчислювальних зусиль, і в результаті можуть знизити загальну продуктивність системи управління.

Налаштування параметрів для алгоритмів обробки зображень також може бути викликом. Кожне зображення є унікальним, що добавляє ще більше викликів до налаштування параметрів для алгоритмів обробки зображень. Різні зображення можуть мати різні характеристики, такі як рівень освітлення, контраст або шум, що вимагає індивідуального підходу до кожного з них. Визначення оптимальних параметрів для кожного зображення це затратний за часом процес, який може знизити продуктивність системи. Також, це створює необхідність в розробці методів машинного навчання та оптимізації для автоматичного налаштування параметрів, заснованого на властивостях конкретного зображення. [3].

Системи технічного зору продовжують розвиватись, перетворюючи не тільки промисловий сектор, але й сферу безпеки, медицини та інше. Відповідно, продовження досліджень в області обробки зображень та впровадження новітніх технологій є вкрай важливим для подальшого покращення цих систем. Особлива увага при цьому приділяється подоланню викликів, пов'язаних з шумом на зображеннях, складністю обробки та налаштуванням параметрів.

# 2 МЕТОДИ ТА ЗАСОБИ ОТРИМАННЯ ВІДЕОДАНИХ

## 2.1 Завантаження відео з файлу

Для роботи з відеоданими, є важливо знати, як завантажувати відео з файлу. Більшість сучасних програмувальних мов та фреймворків мають інструменти та бібліотеки, які допомагають в цьому процесі.

Загальний процес включає декілька кроків. По-перше, потрібно вибрати файл, з якого буде завантажено відео. Цей файл може бути будь-якого типу, що підтримує відео, такого як .mp4, .avi, .mov тощо.

Наступний крок - це відкриття файлу за допомогою відповідного інструменту або бібліотеки. При цьому, як правило, виконується читання метаданих відео, таких як довжина, роздільна здатність, кадрова частота тощо.

Після цього відео завантажується в пам'ять, де воно може бути оброблене або проаналізоване. Деякі системи дозволяють вам одразу працювати з відео, виконуючи обробку в реальному часі, тоді як інші вимагають повного завантаження відео до початку обробки.

Важливо зауважити, що завантаження відео з файлу може бути досить ресурсозатратним процесом, особливо при роботі з великими відеофайлами. Оптимізація цього процесу може бути однією з ключових задач при розробці систем обробки відео [2].

Завантаження відео з файлу на мові Python (лістинг А.1) реалзіується за допомогою бібліотеки обробки зображень OpenCV.

По-перше, імпортуються необхідні бібліотеки: OpenCV (cv2) для роботи з відео та os.path для оперування шляхами до файлів.

В константах задається шлях до відеофайлу (VIDEO\_PATH)

Функція main() містить основний код програми. Спочатку виконується зчитування вводу від користувача для вибору джерела відео: відеофайл або веб-камера.

Потім створюється об'єкт cv2.VideoCapture, який представляє джерело відео. Його аргументом є шлях до відеофайлу

Далі відео проходить через цикл, де виконується зчитування кадрів за допомогою методу cap.read(). Цей метод повертає два значення: булеве значення, що вказує, чи було зчитування кадру успішним, та сам кадр.

Якщо зчитування було невдалим, виводиться повідомлення про помилку та цикл переривається.

Якщо зчитування було успішним, кадр відображається у вікні OpenCV за допомогою методу cv2.imshow(). Цикл продовжується, поки користувач не натисне клавішу 'q'.

Адреси ресурсів, пов'язаних з джерелом відео, звільняються за допомогою методу cap.release(), а вікна OpenCV закриваються за допомогою методу cv2.destroyAllWindows(). На рисунку B.1 зображено екрану форму виконання завантаження відео з файлу.

## 2.2 Захват відео з веб-камери

Захоплення відео з веб-камери - це процес отримання послідовності зображень (або відеопотоку) від веб-камери, що підключена до комп'ютера. Це можна робити для реального часу відеострімінгу, відеочату, відеоконференцій, створення відео, збору відеоданих для обробки та аналізу та інших застосунків.

Використання веб-камери для захоплення відео може залежати від різних факторів, таких як наявність та сумісність веб-камери, використовуване програмне забезпечення, тип операційної системи тощо. Це також вимагає доступу до апаратних ресурсів та дозволів.

У багатьох сучасних веб-камерах є вбудовані датчики, які можуть надавати допоміжні дані, такі як розкішність, кадрову частоту та освітленість. Ці датчики можуть бути корисними для налаштування параметрів відеозахоплення та покращення якості відео [2].

Захват відео з веб-камери на мові Python (лістинг А.1) реалзіується аналогічно завантаженню відео з файлу, проте в параметри конструктора cv2.VideoCapture передається не шлях до файлу а індекс веб-камери, якщо пристрій має одну веб-камеру то індекс за замовчуванням дорівнює 0. На рисунку B.2 зображено екрану форму захвату відео з веб-камери.

# 3 КОЛІРНІ ПЕРЕТВОРЕННЯ ВІДЕО-ЗОБРАЖЕНЬ

## 3.1 Можливості бібліотеки OpenCV з колірних перетворень

Однією з основних можливостей OpenCV є можливість виконувати колірні перетворення на зображеннях. Колірне перетворення - це процес зміни просторового представлення кольору зображення.

Один з найчастіше використовуваних колірних перетворень - це перехід від RGB (червоний, зелений, синій) до Grayscale (відтінки сірого) і навпаки. Це перетворення часто використовується при обробці зображень, особливо при вирішенні проблеми розпізнавання об'єктів, оскільки воно зменшує кількість даних, які потрібно обробити.

Відоме перетворення з RGB до HSV (Hue, Saturation, Value - відтінок, насиченість, яскравість) дозволяє нам розділити "інформацію про кольори" (відтінок) і "інформацію про яскравість" (яскравість).

OpenCV також має функціональність для виконання більш складних колірних перетворень, таких як перетворення від sRGB до LAB або YCrCb. Це може бути корисним при різних проблемах обробки зображень, наприклад, при корекції кольору або коли потрібно максимально зберегти відтінки кольорів [5].

Використовуючи OpenCV, ви можете виконувати колірне перетворення зображень, використовуючи просту функцію cv2.cvtColor(). Ця функція приймає два аргументи: зображення, яке ви хочете перетворити, і цільову колірну схему [6].

Отже, загалом, OpenCV має потужні можливості для виконання колірних перетворень на зображеннях, що може бути корисним для широкого спектра застосунків обробки зображень.

## 3.2 Метод зміни колірного перетворення для колірного простору HSV

HSV - це колірний простір, який поділяється на три компоненти:

Hue (відтінок): це кутова компонента, в якій кожен колір представлений певним кутом. Відтінки знаходяться у діапазоні від 0 до 180 в OpenCV.

Saturation (насиченість): вимірює ступінь відхилення білого кольору від чистого кольору (ненасичений відтінок). Насиченість знаходиться в діапазоні від 0 до 255.

Value (яскравість): відповідає за яскравість кольору. Значення також знаходиться в діапазоні від 0 до 255.

При конвертації з BGR до HSV, кожна компонента BGR кольору переводиться в соответсвующу компоненту HSV за допомогою наступних математичних виразів [1]:

Очевидно, що H обчислюється трохи інакше аніж V та S, в залежності від того, яка компонента RGB має максимальне значення.

Зверніть увагу, що кут відтінку H потім масштабується, щоб він лежав в діапазоні, що підтримується OpenCV (0-180).

## 3.3 Реалізація на Python й аналіз результатів

Для реалізації на Python (лістинг A.2) в OpenCV для конвертації зображень між різними колірними просторами використовується функція cv2.cvtColor(). Щоб змінити колірне простору зображення на HSV (Hue, Saturation, Value), використовується ця функція з константою cv2.COLOR\_BGR2HSV як аргумент. На рисунку B.3 зображено екрану форму зміни колірного простору на HSV. Аналізуючи результат очевидно отримуємо змінене відео зображення.

Функція cv2.cvtColor() забезпечує зручний спосіб перетворення зображень з одного колірного простору до іншого. Використання HSV може бути корисним для роботи з кольорами на більш інтуїтивно зрозумілому для людського сприйняття рівні.

# 4 ГЕОМЕТРИЧНІ ПЕРЕТВОРЕННЯ ВІДЕО-ЗОБРАЖЕНЬ

## 4.1 Можливості бібліотеки OpenCV з геометричних перетворень

Бібліотека OpenCV надає основні та складні інструменти для виконання геометричних перетворень на зображеннях. Геометричні перетворення змінюють просторову конфігурацію зображення, але не змінюють колірні властивості пікселів [5].

Основні геометричні перетворення в OpenCV включають перенесення, масштабування, обертання та викривлення.

Перенесення: це простіший вид геометричного перетворення. Він зсуває положення об'єкта. В OpenCV ви можете використовувати функцію cv2.warpAffine для створення матриці трансформації і застосування її до зображення.

Масштабування: це перетворення, яке змінює розмір зображення. OpenCV надає функцію cv2.resize() для масштабування зображень.

Обертання: Це перетворює зображення навколо певної точки. В OpenCV, ви можете створити матрицю обертання за допомогою функції cv2.getRotationMatrix2D, а потім застосувати її до зображення за допомогою cv2.warpAffine [6].

Викривлення (або спотворення): Це перетворює зображення таким чином, що лінії, які були прямі в оригіналі, можуть стати кривими в результаті. Це може використовуватіться для корекції спотворень лінзи або для створення спеціальних ефектів.

Підтримуються також більш складні геометричні перетворення, такі які афінні та перспективні перетворення. Афінні перетворення зберігають колінеарність та пропорції, тоді як перспективні перетворення, які симулюють перспективу 3D, не зберігають ці властивості [5].

## 4.2 Метод геометричного перетворення для перенесення.

В OpenCV, виконання геометричного перенесення на зображенні здійснюється через функцію cv2.warpAffine() [6]. Для перенесення потрібно створити матрицю трансформації, яка дозволить вам визначити вектор зсуву по осях x та y. Матричне представлення геометричного перенесення базується на однорідних координатах, які додають додатковий розмір для керування трансформаціями. Це дає змогу виконувати складні трансформації, такі як обертання та масштабування, використовуючи множення матриць.

Інверсні (або обернені) перенесення також можуть бути здійснені з використанням знаку мінус з значеннями зсуву по вертикалі та горизонталі для пересування зображення у відповідних напрямках.

Якщо зсув виводить деякі пікселі за межі зображення, ви можете визначити, як повинні бути заповнені ці межі, використовуючи різні методи (наприклад, використовуючи дублювання або відображення) [3].

## 4.3 Реалізація на Python й аналіз результатів

В реалізації на Python (лістинг А.3) використовується OpenCV для перенесення зображення, отриманого від відеофайлу або веб-камери.

У кожному кадрі відео проводиться перенесення. Функція parallel\_transfer створює матрицю перенесення з використанням вхідних параметрів left та top, які вказують на відстань зсуву по горизонталі та вертикалі відповідно. Потім розрахована матриця перенесення застосовується до кадру за допомогою функції cv2.warpAffine.

Отриманий зсунутий кадр відображається в окремому вікні (рисунок Б.4).

Такий код може бути використаний для "переміщення" області видимості відео або перенесення зображення, з камери в реальному часі. Слід зауважити, що при великих значеннях зсуву частина зображення, що виходить за його межі, буде втрачена.

# 5 ОПЕРАЦІЇ З ВІДЕО-ЗОБРАЖЕННЯМИ

## 5.1 Можливості бібліотеки OpenCV для виконання операцій з відеозображеннями

OpenCV надає не тільки інструменти для зміни кольорового простору, геометричних трансформацій та фільтрації зображень, але й безліч інших інструментів для обробки відеозображень такі як:

Відстеження об'єктів, що включає в себе кілька алгоритмів відстеження об'єктів, що дозволяють ідентифікувати та слідкувати за об'єктами на протязі відеопотоку.

OpenCV містить навчені моделі для детекції обличчя та розпізнавання особливостей обличчя. Ці інструменти можна використовувати для розпізнавання обличчя в реальному часі.

OpenCV має відмінні інструменти для виявлення країв та кутів на зображенні, такі як фільтри Sobel і Canny.

OpenCV включає декілька алгоритмів, таких як SIFT, SURF, ORB та многих інших, для виявлення ключових точок та обчислення дескрипторів, які кодують унікальні властивості цих точок.

Віртуальна реальність та 3D візуалізація. Це включає моделювання епіполярної геометрії, відновлення глибини за допомогою стереокарт, а також роботу з об'єктами та сценами Point Cloud [5].

Ці та інші особливості роблять OpenCV ефективним та гнучким інструментом для обробки відеозображень, який можна використовувати в широкому діапазоні застосувань, включаючи системи безпеки, автономні транспортні засоби, медичну зображувальну діагностику, промислову автоматизацію та багато інших.

## 5.2 Метод операції з відео-зображеннями для виділення меж з різними порогами canny.

Виявлення меж з використанням алгоритму Canny - це популярний метод для визначення меж об'єктів на зображенні або відео. Він був названий на честь Джона Кенні, який розробив його в 1986 році. [2] Цей алгоритм складається з кількох етапів:

Спрямоване згладжування (Gaussian smoothing): Зображення спочатку згладжується за допомогою гауссового фільтра для видалення шуму. Це робиться для того, щоб уникнути помилкового виявлення шуму як межі.

Виявлення градієнтів: Після цього силу і напрям градієнтів розраховується. Фактично це представляє інтенсивність зміни яскравості за допомогою часткових похідних по осях x і y.

Немаксимальне придушення (Non-maximum suppression): Цей крок допомагає відокремити "справжні" границі від "псевдограниць". Воно працює шляхом перевірки, чи піксель є локальним максимумом у вказаному напрямі.

Використання двох порогових значень для визначення справжніх меж:

* сильних меж, які мають інтенсивність градієнта вище високого порогу;
* слабкі межі, які мають інтенсивність градієнта нижче високого порогу, але вище низького порогу.

Якщо інтенсивність межі менше низького порогу, то вона відкидається. Якщо вона знаходиться між низьким і високим порогами, вона вважається границею тільки в тому випадку, якщо вона з'єднана з "сильною" межею [3].

## 5.3 Реалізація на Python й аналіз результатів

Даний код (лістинг A.4) використовує бібліотеку OpenCV для детектування меж на відеозображенні за допомогою алгоритму Canny. Відео може бути зчитане з файлу або веб-камери.

Ключова функція у цьому коді - це canny\_edge\_detection(). Вона приймає кадр відео, перетворює його в чорно-біле (або сіре) зображення, а потім використовує детектор меж Canny для визначення меж на цьому зображенні. За замовчуванням для цієї детекції використовуються порогові значення 100 та 200, але це можна змінити засобами передачі інших значень в показниках t\_lower та t\_upper.

Для кожного кадру виконується детекція меж за допомогою функції canny\_edge\_detection(), та результати відображаються на екрані.

Алгоритм Canny хороший для виявлення меж, оскільки він використовує різні пороги для виявлення слабких та сильних границь. У даному скрипті використовуються пороги 100 та 200, що роблять його чутливим до границь середньої та високої інтенсивності. Результатом є чорно-біле зображення, де білі лінії вказують на виявлені межі (рисунок Б.5).

# 6 ФІЛЬТРАЦІЯ ВІДЕО-ЗОБРАЖЕНЬ

## 6.1 Можливості бібліотеки OpenCV з фільтрації

Бібліотека OpenCV включає в себе широкий спектр функцій, які допомагають вирішувати різноманітні задачі, включаючи фільтрацію зображень.

Фільтрація є інтегральною частиною обробки зображень. Більшість операторів обробки зображень, особливо тих, які моделюють людське бачення, базуються на фільтрах. Ось деякі можливості OpenCV щодо фільтрації:

Одним з широко використовуваних методів для зменшення шуму та деталізації зображення є гауссове розмиття (Gaussian blur).

Середнє розмиття (Average blur). Цей фільтр працює шляхом заміни кожного пікселя середнім значенням всіх пікселів навколо. Це допомагає згладити текстури на зображеннях та зменшити шум.

Медіанне розмиття (Median blur). Цей метод замінює кожен піксель медіаною всіх пікселів у радіусі навколо вхідного пікселя. Це допомагає видалити шум, зберігаючи при цьому краї.

Фільтр bilateral (двосторонній): Двостороннє розмиття в галузі комп'ютерного бачення - це неординарний фільтр, який зменшує шум, зберігаючи при цьому краї.

Морфологічні фільтри такі як дилатація, ерозія, відкриття та закриття - це прості способи змінити форму, структуру або текстуру зображення.

Фільтри Лапласа і Собеля: Це оператори диференціювання, які використовуються для виявлення граней чи країв зображень [12.

Ці та багато інших функцій OpenCV роблять його вкрай ефективним інструментом для інженерії характеристик, передобробки зображень, підвищення якості та видалення шумів, а також для виявлення країв та інших подібних задач обробки зображень.

## 6.2 Метод обробки відео-зображень для фільтрації з ефектом зсуву з різними параметрами

Фільтр з ефектом зсуву — це тип обробки зображень, який "переміщує" пікселі в зображенні в певному напрямку. Цей ефект можна використовувати для створення художнього ефекту руху або для візуального вираження швидкості та напрямку об'єктів на зображенні. Базовий алгоритм для фільтрації з ефектом зсуву включає введення значень зсуву по горизонталі (dx) або віртикалі (dy). Фільтр зсуву — це досить простий виклик фунції, тому в ньому немає вбудованого алгоритму чи математичного апарату для розуміння. Основна заслуга тут — це знання того, як матриця трансформації впливає на зображення та як ми можемо це використовувати для досягнення бажаного ефекту [3].

## 6.3 Реалізація на Python й аналіз результатів

Реалізація фільтрації шумів з ефектом зсуву з різними параметрами представлена у лістингу A.5. Основна функція, що використовується в цьому коді, — це motion\_blur(). Ця функція приймає кадр відео та розмір ядра, створює матрицю ядра для зсуву в одному напрямку, а потім використовує функцію cv2.filter2D() для застосування цього ефекту до кадру. Функція cv2.filter2D() здійснює операцію згортки між вхідним зображенням та ядром.

Цей скрипт реалізує фільтр з доволі простим ефектом зсуву. Все, що він робить, — це "зсуває" пікселі зображення вліво або вправо. Результатом є зображення з видимим ефектом руху (рисунок Б.6). Розмір ядра визначає "силу" цього ефекту. У даному випадку, ядро розміром 5 використовується, що забезпечує помірний ефект зсуву.

Однак, слід зазначити, що цей фільтр є досить примітивним і не придатний для великої кількості потреб, що можуть виникнути при роботі з відео. Ефекти зсуву можуть бути набагато більш складними і потребувати використання більш потужних алгоритмів та математичних апаратів.

# 7 РЕАЛІЗАЦІЯ КЛАСУ ДЛЯ ОБРОБКИ ВІДЕОДАНИХ

## 7.1 Поняття класу та його створення на Python

У програмуванні клас — це користувацьки визначаємий прототип для об'єкта, що визначає набір атрибутів, що характеризують об'єкт класу. Атрибути — це дані по членах (які називаються поля) і методи, доступні для об'єкта певного класу.

У мові програмування Python класи створюються за допомогою ключового слова class. Класи організують код навколо об'єктно-орієнтованого підходу, об'єднуючи властивості (характеристики) та поведінки (функції або методи). Зазвичай представник визначеного класу називається екземпляром класу.

Однією з ключових особливостей класів в Python є наявність вбудованих спеціальних методів, таких як \_\_init\_\_. Метод \_\_init\_\_ використовується для ініціалізації атрибутів об'єкта під час створення.

Атрибути та методи є фундаментальними поняттями об'єктно-орієнтованого програмування в Python.

Методи — це функції, які визначають поведінку об'єктів класу. Вони використовуються для зміни стану об'єкта або для взаємодії між об'єктами. Методи викликаються за допомогою крапки (.) після імені об'єкта. [7]

## 7.2 Реалізація класу для обробки відеоданих VideoProcessor

VideoProcessor (лістинг А.6) – цe клас, визначений для обробки відеофайлів. Клас може виконати чотири різні типи обробки в залежності від атрибуту mode, кожен з яких розглядає кадри відео за допомогою різних оброблювальних методів. Також клас має можливість робити зчитування відеоданих з вебкамер або відеофайлів.

Атрибути класу:

MOTION\_BLUR, CHANGE\_COLORSPACE, CANNY\_EDGE\_DETECTION, TRANSFER (int) – Константи, що визначають можливі режими обробки, що повинні використовуватись класом.

\_\_video\_source (int, str) - Джерело відео-зображення, яке може бути індексом веб-камери або шляхом до відеофайлу.

\_\_mode (int) - Поточний обраний режим обробки.

Методи класу:

\_\_init\_\_(self, video\_source: Union[int, str] ) - Конструктор класу, який ініціалізує вихідне відео та режим обробки.

set\_mode(self, mode: int )- Метод для встановлення режиму обробки. Метод перевіряє, чи є режим обробки валідним.

set\_video\_source(self, video\_source: Union[int, str]) - Метод для встановлення вихідного відео. Метод перевіряє, чи є вихідне джерело валідним.

\_\_motion\_blur(self, frame: MatLike, kernel\_size: Optional[int] = 5) - Функція для створення ефекту зсуву зображення.

\_\_change\_colorspace(self, frame: MatLike) - Функція для зміни кольорового простору зображення на HSV.

\_\_canny\_edge\_detection(self, image: MatLike, t\_lower: Optional[int] = 100, t\_upper: Optional[int] = 200) - Функція для виділення виділення меж з різними порогами.

\_\_parallel\_transfer(self, frame: MatLike, left: Optional[int] = 200, top: Optional[int] = 150) - Функція для переносу зображення

process(self, kernel\_size: Optional[int] = 5, left: Optional[int] = 200, top: Optional[int] = 150, high: Optional[int] = 100, low: Optional[int] = 200) - Метод для обробки відео даних згідно з обраним режимом.

На рисунку Б.7 продемонстровано використання класу для обробки відеоданих з файлу або відеокамери.

# Висновки

В ході виконання роботи дослідженно різноманітні методи обробки зображень та відео, які вклюють в себе такі методи обробки відео, як захвату з відео-камери відео-зображень, колірні перетворення відео-зображення (колірний простір HSV), геометричні перетворення відео-зображення (перенесення), виконання операцій над відео-зображенням (виділення меж з різними порогами canny), фільтрація шумів відео-зображення (фільтр з ефектом зсуву з різними параметрами). Всі ці методи надають унікальні можливості для різних сценаріїв обробки відео залежно від конкретних вимог до проекту.

Описано реалізація класу для обробки відеоданих з файлу або відеокамери. Клас дозволяє проводити різні типи обробки кадрів відео, зокрема ефект розмиття руху, роботу з кольоровим простором, виділення меж за допомогою розроблених методів. Було проведено успішне вивчення та практичне застосування принципів об'єктно-орієнтованого програмування в Python. Розроблений клас слугує прикладом того, як можна інкапсулювати різні алгоритми обробки відео в межі одного класу, що забезпечує його читабельність та можливість повторного використання.

Наприкінці, варто виділити значимість обробки відеоданих у сучасному інформаційному суспільстві. Дані у відео форматі відіграють важливу роль, є широко розповсюдженими та особливо значущими. Вони знаходять широке застосування у різних сферах, наприклад, безпека, рекламна індустрія, створення медіа-контенту, комп'ютерний зір, аналіз спортивних дій, визначення та ідентифікація об'єктів та осіб, та багато іншого. Більш того, завдяки доступності та потужності інструментів таких як OpenCV, фахівці та студенти мають можливість з легкістю вивчати та розробляти проекти в цій динамічно розвиваючій сфері.

# Перелік використаних джерел

1. Computer Vision [Текст]: L. Shapiro and G. Stockman – Englewood Cliffs, NJ, Prentice Hall, 2001.
2. Digital Image Processing [Текст]: R. Gonzalez and R. Woods – Pearson, 2008
3. Two-Dimensional Signal and Image Processing [Текст]: J. S. Lim – Englewood Cliffs, NJ, Prentice Hall, 1990
4. Основы построения современных мобильных систем технического зрения [Текст]: учеб. пособие (часть 2). / Л. А. Краснов, К. Ю. Дергачев, С. В. Багинский – Х.: Нац. аэрокосм. ун-т им. Н. Е. Жуковского «Харьк. авиац. ин-т», 2018. – 92 с.
5. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library [Текст]: Adrian Kaehler, Gary Bradski
6. Електроний ресурс: http://docs.opencv.org – Документація бібліотеки OpenCV.
7. Електронний ресурс: https://docs.python.org/3/tutorial/classes.html - Офіційний туторіал Python про класи, їх атрибути та методи.

# Додаток А

Лістинг коду

А.1 Лістинг коду для зчитування відео з файлу та захвату відео з вебкамери

# Підключення необхідних бібліотек

import cv2 # OpenCV

from os import path # Для шляху до файлу

# Константи

VIDEO\_PATH = path.join("data", "Lion.mp4") # Шлях до відео

CAMERA\_INDEX = 0 # Індекс веб-камери

def main():

while True:

# Вибір джерела відео

print("Оберіть джерело відео:")

print("1. Відеофайл")

print("2. Вебкамера")

choise = input("Введіть номер (1 або 2): ")

# Перевірка правильності вводу

if choise in ["1", "2"]:

choise = int(choise)

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

# Метод зчитування даних з відеофайлу/вебкамери

cap = cv2.VideoCapture(VIDEO\_PATH if choise == 1 else CAMERA\_INDEX)

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Помилка запису фрейму!")

break

# Відображення результату

cv2.imshow("frame", frame)

# Натискання клавіші "q" для виходу з циклу

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Звільнення ресурсів, пов'язаних із відеофайлом

cap.release()

# Закриття всіх вікон OpenCV

cv2.destroyAllWindows()

# При запуску як головного файлу

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

main()

А.2 Лістинг коду для колірного перетворення відео-зображення (колірний простір HSV)

# Підключення необхідних бібліотек

import cv2 # OpenCV

from os import path # Для шляху до файлу

# Константи

VIDEO\_PATH = path.join("data", "Lion.mp4") # Шлях до відео

CAMERA\_INDEX = 0 # Індекс веб-камери

def main():

while True:

# Вибір джерела відео

print("Оберіть джерело відео:")

print("1. Відеофайл")

print("2. Вебкамера")

choise = input("Введіть номер (1 або 2): ")

# Перевірка правильності вводу

if choise in ["1", "2"]:

choise = int(choise)

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

# Метод зчитування даних з відеофайлу/вебкамери

cap = cv2.VideoCapture(VIDEO\_PATH if choise == 1 else CAMERA\_INDEX)

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Помилка запису фрейму!")

break

# Зміна колірного простору зображення (фрейму) на HSV

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

# Відображення результату

cv2.imshow("frame", frame)

# Натискання клавіші "q" для виходу з циклу

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Звільнення ресурсів, пов'язаних із відеофайлом

cap.release()

# Закриття всіх вікон OpenCV

cv2.destroyAllWindows()

# При запуску як головного файлу

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

main()

А.3 Лістинг коду для геометричного перетворення відео-зображення (перенесення)

# Підключення необхідних бібліотек

import cv2 # OpenCV

from os import path # Для шляху до файлу

from numpy import float32 # Для роботи з матрицею

# Константи

VIDEO\_PATH = path.join("data", "Lion.mp4") # Шлях до відео

CAMERA\_INDEX = 0 # Індекс веб-камери

# Функція переносу зображення

def parallel\_transfer(frame, left, top):

# Отримання кількості рядків та стовпців у вихідному кадрі

num\_rows, num\_cols = frame.shape[:2]

# Створення матриці трансляції для здійснення зсуву кадру, де left - зсув по горизонталі, а top - зсув по вертикалі

translation\_matrix = float32([[1, 0, left], [0, 1, top]])

# Перенос зображення

transfered\_frame = cv2.warpAffine(frame, translation\_matrix, (num\_cols, num\_rows))

return transfered\_frame

def main():

while True:

# Вибір джерела відео

print("Оберіть джерело відео:")

print("1. Відеофайл")

print("2. Вебкамера")

choise = input("Введіть номер (1 або 2): ")

# Перевірка правильності вводу

if choise in ["1", "2"]:

choise = int(choise)

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

# Метод зчитування даних з відеофайлу/вебкамери

cap = cv2.VideoCapture(VIDEO\_PATH if choise == 1 else CAMERA\_INDEX)

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Помилка запису фрейму!")

break

# Перенесення кадру на 180 по горизонталі та 320 по вертикалі

frame = parallel\_transfer(frame, 180, 320)

# Відображення результату

cv2.imshow("frame", frame)

# Натискання клавіші "q" для виходу з циклу

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Звільнення ресурсів, пов'язаних із відеофайлом

cap.release()

# Закриття всіх вікон OpenCV

cv2.destroyAllWindows()

# При запуску як головного файлу

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

main()

А.4 Лістинг коду для виконання операцій над відео-зображенням (виділення меж з різними порогами canny)

# Підключення необхідних бібліотек

import cv2 # OpenCV

from os import path # Для шляху до файлу

# Константи

VIDEO\_PATH = path.join("data", "Lion.mp4") # Шлях до відео

CAMERA\_INDEX = 0 # Індекс веб-камери

# Функція виділення виділення меж з різними порогами canny

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

# Перетворення кадру в колірний простір GRAY

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Виділення виділення меж з різними порогами canny

edge = cv2.Canny(gray\_image, t\_lower, t\_upper)

return edge

def main():

while True:

# Вибір джерела відео

print("Оберіть джерело відео:")

print("1. Відеофайл")

print("2. Вебкамера")

choise = input("Введіть номер (1 або 2): ")

# Перевірка правильності вводу

if choise in ["1", "2"]:

choise = int(choise)

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

# Метод зчитування даних з відеофайлу/вебкамери

cap = cv2.VideoCapture(VIDEO\_PATH if choise == 1 else CAMERA\_INDEX)

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Помилка запису фрейму!")

break

# Виділення виділення меж з різними порогами low = 100 top=200

frame = canny\_edge\_detection(frame, 100, 200)

# Відображення результату

cv2.imshow("frame", frame)

# Натискання клавіші "q" для виходу з циклу

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Звільнення ресурсів, пов'язаних із відеофайлом

cap.release()

# Закриття всіх вікон OpenCV

cv2.destroyAllWindows()

# При запуску як головного файлу

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

main()

А.5 Лістинг коду для фільтрації шумів відео-зображення (фільтр з ефектом зсуву з різними параметрами)

# Підключення необхідних бібліотек

import cv2 # OpenCV

from os import path # Для шляху до файлу

from numpy import zeros, ones # Для створення матриці

# Константи

VIDEO\_PATH = path.join("data", "Lion.mp4") # Шлях до відео

CAMERA\_INDEX = 0 # Індекс веб-камери

# Фільтрація з ефектом зсуву

def motion\_blur(frame, kernel\_size):

# Створення матриці нулів розміром kernel\_size x kernel\_size для ядра рухового розмиття

kernel\_motion\_blur = zeros((kernel\_size, kernel\_size))

# Встановлення значень одиниць у центральному рядку матриці ядра рухового розмиття

kernel\_motion\_blur[int((kernel\_size - 1) / 2), :] = ones(kernel\_size)

# Нормалізація ядра рухового розмиття, ділення кожного значення на kernel\_size

kernel\_motion\_blur = kernel\_motion\_blur / kernel\_size

# Застосування фільтрації до зображення за допомогою ядра рухового розмиття

return cv2.filter2D(frame, -1, kernel\_motion\_blur)

def main():

while True:

# Вибір джерела відео

print("Оберіть джерело відео:")

print("1. Відеофайл")

print("2. Вебкамера")

choise = input("Введіть номер (1 або 2): ")

# Перевірка правильності вводу

if choise in ["1", "2"]:

choise = int(choise)

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

# Метод зчитування даних з відеофайлу/вебкамери

cap = cv2.VideoCapture(VIDEO\_PATH if choise == 1 else CAMERA\_INDEX)

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Помилка запису фрейму!")

break

# Фільтрація з ефектом зсуву, kernel\_size = 5

frame = motion\_blur(frame, 5)

# Відображення результату

cv2.imshow("frame", frame)

# Натискання клавіші "q" для виходу з циклу

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Звільнення ресурсів, пов'язаних із відеофайлом

cap.release()

# Закриття всіх вікон OpenCV

cv2.destroyAllWindows()

# При запуску як головного файлу

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

main()

А.6 Лістинг коду для реалізації класу для обробки відеоданих з файлу або відеокамери.

# Підключення необхідних бібліотек

import cv2 # OpenCV

from os import path # Для шляху до файлу

from numpy import zeros, ones, float32 # Для створення матриці

# Для вибору файлу з пк

from tkinter import filedialog

class VideoProcessor:

# Константи режимів обробки відео

MOTION\_BLUR = 1

CHANGE\_COLORSPACE = 2

CANNY\_EDGE\_DETECTION = 3

TRANSFER = 4

# Конструктор

def \_\_init\_\_(self, video\_source):

self.\_\_video\_source = 0

self.\_\_mode = self.MOTION\_BLUR

self.set\_video\_source(video\_source)

def set\_mode(self, mode):

# Перевірка на валідність переданого режиму

if mode not in [self.MOTION\_BLUR, self.CHANGE\_COLORSPACE, self.CANNY\_EDGE\_DETECTION, self.TRANSFER]:

raise ValueError("Неправильний режим обробки відео")

self.\_\_mode = mode

def set\_video\_source(self, video\_source):

if isinstance(video\_source, int):

self.\_\_video\_source = video\_source

return

# Перевірка чи файл існує

if not path.isfile(video\_source):

raise FileNotFoundError("Файл відео не існує")

# Перевірка розширення файлу

\_, ext = path.splitext(video\_source)

if ext.lower() not in ['.avi', '.mp4', '.mov', '.mkv']:

raise ValueError("Непідтримуваний формат відео")

self.\_\_video\_source = video\_source

# Фільтрація з ефектом зсуву

def \_\_motion\_blur(self, frame, kernel\_size):

# Створення матриці нулів розміром kernel\_size x kernel\_size для ядра рухового розмиття

kernel\_motion\_blur = zeros((kernel\_size, kernel\_size))

# Встановлення значень одиниць у центральному рядку матриці ядра рухового розмиття

kernel\_motion\_blur[int((kernel\_size - 1) / 2), :] = ones(kernel\_size)

# Нормалізація ядра рухового розмиття, ділення кожного значення на kernel\_size

kernel\_motion\_blur = kernel\_motion\_blur / kernel\_size

# Застосування фільтрації до зображення за допомогою ядра рухового розмиття

return cv2.filter2D(frame, -1, kernel\_motion\_blur)

# Зміна кольорового простору на HSV

def \_\_change\_colorspace(self, frame):

return cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

# Функція виділення виділення меж з різними порогами canny

def \_\_canny\_edge\_detection(self, image, t\_lower=100, t\_upper=200):

# Перетворення кадру в колірний простір GRAY

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Виділення виділення меж з різними порогами canny

edge = cv2.Canny(gray\_image, t\_lower, t\_upper)

return edge

# Функція переносу зображення

def \_\_parallel\_transfer(self, frame, left, top):

# Отримання кількості рядків та стовпців у вихідному кадрі

num\_rows, num\_cols = frame.shape[:2]

# Створення матриці трансляції для здійснення зсуву кадру, де left - зсув по горизонталі, а top - зсув по вертикалі

translation\_matrix = float32([[1, 0, left], [0, 1, top]])

# Перенос зображення

transfered\_frame = cv2.warpAffine(frame, translation\_matrix, (num\_cols, num\_rows))

return transfered\_frame

# Функція для обробки відеоданих

def process(self, kernel\_size=5, left=200, top=150, low = 100):

# Метод зчитування даних з відеофайлу/вебкамери

cap = cv2.VideoCapture(self.\_\_video\_source)

# Перевірка готовності веб-камери

while cap.isOpened():

# Запис фреймів

ret, frame = cap.read()

# При виникненні помилці запису

if not ret:

print("Помилка запису фрейму!")

break

# Обробка кадру в залежності від обраного режиму

if self.\_\_mode == self.CHANGE\_COLORSPACE:

frame = self.\_\_change\_colorspace(frame)

if self.\_\_mode == self.MOTION\_BLUR:

frame = self.\_\_motion\_blur(frame, kernel\_size)

if self.\_\_mode == self.CANNY\_EDGE\_DETECTION:

frame = self.\_\_canny\_edge\_detection(frame, low, top)

if self.\_\_mode == self.TRANSFER:

frame = self.\_\_parallel\_transfer(frame, left, top)

# Відображення результату

cv2.imshow("frame", frame)

# Натискання клавіші "q" для виходу з циклу

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Звільнення ресурсів, пов'язаних із відеофайлом

cap.release()

# Закриття всіх вікон OpenCV

cv2.destroyAllWindows()

def select\_file():

return filedialog.askopenfilename(filetypes=[("Video files", "\*.mp4 \*.avi \*.mov \*.mkv")])

def main():

choise = 0

print("Програма для обробки відео")

while True:

# Вибір джерела відео

print("Оберіть джерело відео:")

print("1. Відеофайл")

print("2. Вебкамера")

choise = input("Введіть номер (1 або 2): ")

# Перевірка правильності вводу

if choise in ["1", "2"]:

choise = int(choise)

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

video\_source = 0

if choise == 1:

video\_source = select\_file()

video\_processor = None

try:

video\_processor = VideoProcessor(video\_source)

except ValueError as error:

print(repr(error))

return

choise = 0

while True:

# Вибір режиму обробки відео

print("Оберіть режим обробки:")

print("1. Фільтр з ефектом зсуву")

print("2. Зміна кольорового простору на HSV")

print("3. Виділення меж порогами Canny")

print("4. Перенесення")

choise = input("Введіть номер (від 1 до 4): ")

# Перевірка правильності вводу

try:

choise = int(choise)

except:

print("Некоректний ввід. Спробуйте ще раз.")

continue

if choise in [video\_processor.CHANGE\_COLORSPACE, video\_processor.MOTION\_BLUR, video\_processor.CANNY\_EDGE\_DETECTION, video\_processor.TRANSFER]:

break

else:

print("Некоректний ввід. Спробуйте ще раз.")

print(choise)

if choise == video\_processor.MOTION\_BLUR:

kernel\_size = 0

while True:

kernel\_size = int(input("Введіть розмір ядра (число): "))

try:

kernel\_size = int(kernel\_size)

except:

print("Некоректний ввід. Спробуйте ще раз.")

continue

break

video\_processor.set\_mode(video\_processor.MOTION\_BLUR)

video\_processor.process(kernel\_size=kernel\_size)

if choise == video\_processor.CHANGE\_COLORSPACE:

video\_processor.set\_mode(video\_processor.CHANGE\_COLORSPACE)

video\_processor.process()

if choise == video\_processor.CANNY\_EDGE\_DETECTION:

top = 0

while True:

top = int(input("Введіть верхню межу: "))

try:

top = int(top)

except:

print("Некоректний ввід. Спробуйте ще раз.")

continue

break

low = 0

while True:

low = int(input("Введіть нижню межу: "))

try:

low = int(low)

except:

print("Некоректний ввід. Спробуйте ще раз.")

continue

break

video\_processor.set\_mode(video\_processor.CANNY\_EDGE\_DETECTION)

video\_processor.process(low=low, top=top)

if choise == video\_processor.TRANSFER:

left = 0

top = 0

while True:

top = int(input("Введіть зміщення по вертикалі: "))

try:

top = int(top)

except:

print("Некоректний ввід. Спробуйте ще раз.")

continue

break

while True:

left = int(input("Введіть зміщення по горизонталі: "))

try:

left = int(left)

except:

print("Некоректний ввід. Спробуйте ще раз.")

continue

break

video\_processor.set\_mode(video\_processor.TRANSFER)

video\_processor.process(left=left, top=top)

# При запуску як головного файлу

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

main()

# Додаток Б

Екранні форми виконання

Рисунок Б.1 – Завантаження відео з файлу

Рисунок Б.2 – Захват відео з відео-камери

Рисунок Б.3 – Колірні перетворення відео-зображення (колірний простір HSV)

Рисунок Б.4 – Геометричні перетворення відео-зображення (перенесення)

Рисунок Б.5 – Виконання операцій над відео-зображенням (виділення меж з різними порогами canny)

Рисунок Б.6 – Фільтрація шумів відео-зображення (фільтр з ефектом зсуву з різними параметрами)

Рисунок Б.7 – Використання класу для обробки відеоданих з файлу або відеокамери.