РЕФЕРАТ

Листів 39, ілюстрацій 12, джерел 5, додатків 2.

Побудова загального інтерфейсу для обробки відеоданих з використанням об'єктно-орієнтованого підходу.

В даному рефераті міститься 7 основних розділів, в яких виконуються такі операції як кольорові перетворення відео-зображень, геометричні перетворення, різноманітні операції, фільтрація та реалізація класу для обробки відеоданих.

В результаті були отримані дзеркальне відображення відео-зображень, обробка детектуванням кутів Ши-Томасі, змінення кольорового простору та застосування Гаусівського фільтру.

КЛЮЧОВІ СЛОВА: відеодані, фільтрація, обробка, перетворення, лістинг коду, екранні форми перевірки, бібліотека OpenCV, файл, відео-зображення, кадр.

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ ПОЗНАЧЕНЬ

ТЗ – технічний зір;

ПЗ – програмне забезпечення;

ВЗ – відео-зображення;

ВК – веб-камера;

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується розробкою, створенням та широким впровадженням відеоінформаційних технологій, що засновані на обробці та використанні зображень. На даний час актуальність відеоінформаційного напрямку здебільшого обумовлена потребами розвитку штучних інтелектуальних систем, які повинні мати можливості з візуальної орієнтації у просторі та є придатними до візуального аналізу сцен, візуального пошуку нерухомих та/або рухомих об’єктів з оцінюванням їх геометричних форм й кількісних характеристик. Такі можливості є важливими споживчими рисами для інтелектуальних систем не лише промислового, але й звичайного побутового призначення.

Системи технічного зору(далі ТЗ) знаходять все ширше застосування в різних галузях промисловості, науки і техніки. Їх використовують для контролю якості продукції, управління роботами, розпізнавання об'єктів, навігації та багато іншого. Одним з ключових компонентів ТЗ є модуль обробки відео-зображень(далі ВЗ).

У цьому рефераті буде розглянуто різні методи обробки ВЗ, які використовуються в системах ТЗ. Буде дано опис основних методів, а також приклади застосування.

1 ОГЛЯД ПРОБЛЕМИ ОБРОБКИ ЗОБРАЖЕНЬ В СИСТЕМАХ УПРАВЛІННЯ З ТЕХНІЧНИМ ЗОРОМ

У практиці використання технічного зору, комп'ютери попередньо запрограмовані для вирішення конкретних завдань, але методи їх вирішення, стають все більш узагальненими. В результаті чого, ТЗ іноді розглядають як частину штучного інтелекту або область комп'ютерних наук в цілому.

Як окрема галузь науки, ТЗ займається вивченням методів і технологій створення багатофункціональних штучних систем, які можуть виробляти виявлення, стеження і класифікацію різних заданих об'єктів, отримуючи інформацію із зображень. Для отримання інформації система використовує відеодані, які в свою чергу можуть бути передані для обробки за допомогою різних форм, таких як відеопослідовність, просте зображення або зображення з тривимірними даними.

Основну частину в процесі роботи систем ТЗ займає автоматичне планування, постановка задач та їх рішень в системах, які можуть виконувати механічні дії (вимірювання положення і орієнтації деталей). Такий тип обробки потребує заздалегідь наданих технічних вихідних даних, так само варто відзначити, що такі системи можуть працювати з застосуванням штучного інтелекту, який в свою чергу використовує розпізнавання і методи машинного навчання. Дані, які необхідно витягти з картинки (фігури, об'єкти текст) виходять після виконання деяких задач розпізнавання, детектування, сегментація, рух, відновлення сцен і зображень. Розпізнавання образів (об'єктів) важливе завдання з інтетефікації об'єктів по його зображенню, образ - це класифікаційна група, яка збіднює кілька об'єктів за кількома ознаками. Розпізнавання використовують для розпізнавання алфавітно-цифрових символів, біометричних даних, голосу, відбитків пальців, обличчя) розпізнавання аудіоданих.

Сучасні системи ТЗ орієнтовано на використання мови програмування Python на базі операційної системи Windows при їх стаціонарному розміщенні. У мобільних системах ТЗ, що базуються на рухомих носіях, зазвичай застосовують одноплатну платформу Raspberry Pi з рекомендованою виробником операційною системою Raspbian. При цьому базовою мовою програмування також є Python.

Основні завдання ТЗ з оброблення зображень і відеоданих при програмуванні на Python прийнято вирішувати з використанням бібліотеки

Pillow(Python Imaging Library), що забезпечує досить повний набір функцій і методів оброблення зображень і відео. Однак найбільш ефективним є спільне використання ресурсів бібліотеки Pillow і бібліотеки OpenCV (Open Source Computer Vision Library).

Зручним є застосування OpenCV разом з Python для створення простих і зрозумілих програм, проведення експериментів і синтезу різних прототипів. Мовою Python доступна практично вся функціональність бібліотеки OpenCV.

2 МЕТОДИ ТА ЗАСОБИ ОТРИМАННЯ ВІДЕОДАНИХ

2.1 Завантаження відео з файлу

Щоб почати роботу з ВЗ, його необхідно ввести в програму й відобразити для візуального перегляду. Необхідно зауважити, що в коді програми потрібно виконати імпорт ресурсів бібліотеки OpenCV. Ця операція буде успішно виконана, якщо перед початком роботи на комп'ютері здійснено підключення бібліотеки. Крім того, потрібне ВЗ необхідно помістити в системну папку Python. Це спрощує процедуру введення (не треба прописувати шлях до файлу зображення) і запобігає виникненню синтаксичних помилок. Слід нагадати, що обов'язково має бути вказаний відповідний формат файлу(jpg, mp4).

Реалізація на Python починається з підключення бібліотеки OpenCV та створення словника з шляхом ВЗ та індексом ВК, для подальшого зручного запису цих даних у параметри. Наступним є створення екземпляру класу *cv2.VideoCapture(),* параметром якого є шлях до ВЗ. Далі, для даного екземпляру класу існує один з методів - *cap.isopened(),* який допомагає знайти даний відео-файл за вказаним шляхом.

У випадку, коли в файлі є якась інформація для обробки, цей метод повертає значення «true», якщо виникає помилка або відео закінчується – «false». Попередня операція виконується безпосередньо у циклі *while,* в якому виконується ще один основний метод – *cap.read(зчитування),* при якому відбується зчитування і який повертає 2 вихідні параметри, це *ret(чи цілісним є кадр, з яким працюють)* та *frame.*

Завершальною є функція *cv2.imshow()* бібліотеки OpenCV, яка виконує відображення ВЗ. Ця функція приймає 2 основні параметри: *'frame'* - титульна інформація даного вікна, в якому буде відображатися кадр та *frame* – сам кадр для відображення.

Після цього додається додатковий цикл з методом *cv2.waitKey(),* який очікує натискання якої-небудь клавіши. Якщо цей метод не додати, то інший метод – *cv2.imshow()* буде просто видалятися. Далі за допомогою методу *cap.release()* потрібно звільнити даний відкритий файл з ВЗ. І завершити всі операції з даним кодом необхідно останнім методом *cv2.destroyAllWindows(),* який закриває усі відкриті вікна для демонстрації.

Лістинг коду для зчитування відео з файлу представлено у додатку А
Екранні форми перевірки представлені у додатку Б

2.2 Захват відео з веб-камери(далі ВК)

У практиці використання систем технічного зору застосовуються два основні режими роботи з відеоданими;

• уведення й оброблення відео в реальному масштабі часу безпосередньо від відеореєстратора;

• аналіз відеоданих з раніше записаного файлу.

Перший режим зазвичай використовують для вирішення завдання керування рухомими об'єктами на основі аналізу даних відеоспостереження сцени, а другий − для перегляду подій, зареєстрованих, наприклад, у системах охорони об'єктів або при аналізі наукової відеоінформації. Для використання однієї з декількох камер необхідно в функції введення вказати цілочисловий аргумент. Якщо камера єдина, то аргумент слід вибирати таким, що дорівнює нулю – *cv2.Video Capture (0)*. Процедура читання відео досить проста, оскільки OpenCV автоматично підтримує різні формати відеоданих (mp4, avi та ін.). Для забезпечення різних варіантів читання використовується об'єкт модуля *cv2.VideoCapture()*. Як аргумент використовується ціле число або ім'я файлу. Цілочисловий аргумент цього модуля є ідентифікатором підключеної до комп'ютера камери, а ім'я файлу використовується для захвату даних записаного раніше відеофайлу. Вбудована web-камера зазвичай має ідентифікатор 0.

Реалізація на Python починається з підключення бібліотеки OpenCV та створення словника з шляхом ВЗ та індексом ВК, для подальшого зручного запису цих даних у параметри. Наступним є створення екземпляру класу *cv2.VideoCapture(),* параметром якого є шлях до ВЗ. Далі, для даного екземпляру класу існує один з методів - *cap.isopened(),* який допомагає знайти даний відео-файл за вказаним шляхом.

У випадку, коли в файлі є якась інформація для обробки, цей метод повертає значення «true», якщо виникає помилка – «false». Попередня операція виконується безпосередньо у циклі *while,* в якому виконується ще один основний метод – *cap.read(зчитування),* при якому відбується зчитування і який повертає 2 вихідні параметри, це *ret(чи цілісним є кадр, з яким працюють)* та *frame*. Ще

додатково можна додати метод *cv2.cvtColor(),* параметром якого є сірий фільтр зі своєю назвою *cv2.COLOR_BGR2GRAY.*

Завершальною є функція *cv2.imshow()* бібліотеки OpenCV, яка виконує показ ВК. Ця функція приймає 2 основні параметри: *'frame'* - титульна інформація даного вікна, в якому буде відображатися кадр та *frame* – сам кадр для відображення.

Після цього додається додатковий цикл з методом *cv2.waitKey(),* який очікує натискання якої-небудь клавіши. Якщо цей метод не додати, то інший метод – *cv2.imshow()* буде просто видалятися. Далі за допомогою методу *cap.release()* потрібно звільнити даний відкритий файл з ВК. І завершити всі операції з даним кодом необхідно останнім методом *cv2.destroyAllWindows(),* який закриває усі відкриті вікна для демонстрації.

Лістинг коду для зчитування відео з веб-камери представлено у додатку А

Екранні форми перевірки представлені у додатку Б

3 ГЕОМЕТРИЧНІ ПЕРЕТВОРЕННЯ ВІДЕО-ЗОБРАЖЕНЬ

3.1 Можливості бібліотеки OpenCV з геометричних перетворень

Поворот, зміна розмірів зображення, зсув та інші перетворення належать до геометричних перетворень, які називаються афінними. Афінне перетворення можна записати у відповідній матричній формі.

3.2 Метод *cv2.warpAffine()* для *виконання дзеркального відображення*

Зміщувати зображення вздовж осей можна робити за допомогою функції *cv2.warpAffine*(). При цьому ми можемо переміщувати картинку вліво та вправо, вниз та вгору, а також будь-яку комбінацію з перерахованого. Функція повертає зміщене зображення, яке має розмір dsize і той самий тип, що й вхідне зображення.

Обов'язковими параметрами є src - вхідне зображення, M - матриця перетворення та dsize - розмір вихідного зображення, який задається у вигляді кортеджу ширини та висоти зображення.

У афінному перетворенні всі паралельні прямі у вхідному зображенні залишаться паралельними у вихідному зображенні. В цьому перетворенні зберігається колінеарність, паралельність, а також співвідношення відстаней між точками. Проте не обов’язково зберігаються відстані та кути. Афінне перетворення можна отримати за допомогою матриці перетворення M. Це матриця трансляції, яка зміщує зображення на вектор (x, y).

Щоб знайти матрицю перетворення, нам потрібні три точки з вхідного зображення та їхні відповідні місця на вихідному зображенні. Потім необхідно використати функцію *cv2.getAffineTransform()*, яка створить матрицю 2x3, яку потрібно передати у параметр M функції *cv2.warpAffine().*

3.3 Реалізація на Python й аналіз результатів

Реалізація на Python починається з підключення бібліотек OpenCV та Numpy, далі створення функції *mirror(),* яка просто приймає зображення і видає результат як віддзеркалене зображення на виході. В цій функції задаємо у матричному вигляді розміри вихідного зображення(по стовбцях та колонках),за допомогою методу *float32* бібліотеки Numpy задаємо значення, щодо точного розташування зображення відносно країв та функції *cv2.getAffineTransform()* бібліотеки OpenCV, яка приймає вхідні дані три пари відповідних точок і виводить 2×3 матрицю афінного перетворення M. В самому кінці основної функції створюємо функцію *cv2.warpAffine*(), яка переміщує зображення вліво та вправо, вниз та вгору, повертаючи готовий результат *img_output.* Наступним є створення екземпляру класу *cv2.VideoCapture(),* параметром якого є шлях до ВЗ та ВК. Далі, для даного екземпляру класу існує один з методів - *cap.isopened(),* який допомагає знайти даний відео-файл за вказаним шляхом.

У випадку, коли в файлі є якась інформація для обробки, цей метод повертає значення «true», якщо виникає помилка або відео закінчується – «false». Попередня операція виконується безпосередньо у циклі *while,* в якому виконується ще один основний метод – *cap.read(зчитування),* при якому відбується зчитування і який повертає 2 вихідні параметри, це *ret(чи цілісним є кадр, з яким працюють)* та *frame*. Далі за допомогою функції *mirror* відповідно перетворюємо ВЗ та ВК в геометричне перетворення за варіантом.

Завершальною є функція *cv2.imshow()* бібліотеки OpenCV, яка виконує відображення ВЗ та ВК. Ця функція приймає 2 основні параметри: *'frame'* - титульна інформація даного вікна, в якому буде відображатися кадр та *frame* – сам кадр для відображення. Та не забуваємо зробити 2 функції, одну для кадру «до» та одну для кадру «після» кольорового перетворення.

Після цього додається додатковий цикл з методом *cv2.waitKey(),* який очікує натискання якої-небудь клавіши. Якщо цей метод не додати, то інший метод – *cv2.imshow()* буде просто видалятися. Далі за допомогою методу *cap.release()* потрібно звільнити даний відкритий файл з ВЗ. І завершити всі операції з даним кодом необхідно останнім методом *cv2.destroyAllWindows()*, який закриває усі відкриті вікна для демонстрації.

Лістинг коду для зчитування відео з файлу та веб-камери(«до» та «після» геометричного перетворення «дзеркальне відображення»)представлено у додатку А

Екранні форми перевірки представлені у додатку Б

4 КОЛЬОРОВІ ПЕРЕТВОРЕННЯ ВІДЕО-ЗОБРАЖЕНЬ

4.1 Можливості бібліотеки OpenCV з кольорових перетворень

При зчитуванні зображень кольоровий простір зберігається у форматі BGR(на відміну від звичайного RGB, в цьому форматі змінені перший та останній канали місцями). Функціонально ця заміна нічого не змінює, кожен кадр ВЗ розділяється на 3 окремі канали, які містять інформацію про кольорові канали синього, зеленого та червоного кольорів.

І з використанням методу *cv2.cvtColor()* бібліотеки OpenCV можна даний кадр переформатувати у інший колірний простір. Кольорових просторів існує декілька(це Gray, XYZ, LAB, YUV, HSV), вони насамперед дозволяють отримати та працювати з іншого роду інформацією.

4.2 Метод *cv2.cvtColor()* для *кольорового перетворення*

Набагато більший практичний інтерес становить процедура підвищення контрастності кольорових зображень, яку можна виконати шляхом перетворення rgb-зображення на формат YUV - це колірна модель, у якій колір подається як три компоненти: яскравість (Y) і дві кольоророзницеві (U і V). На зображенні у форматі YUV еквалізації піддається лише компонента яскравості (Y).

Потім здійснюється обернене перетворення з формату YUV на формат RGB. При цьому баланс кольору зберігається без змін, оскільки кольоророзницеві компоненти U і V перетворенням не піддавалися. Формат YUV просто дозволяє отримувати трохи інші дані, тобто не просто кольорові дані, а там використовується окремий канал яскравості і результат виходить у 2 окремих канали.

4.3 Реалізація на Python й аналіз результатів

Реалізація на Python починається з підключення бібліотеки OpenCV та створення функції *colorspace_change(),* яка приймає тільки 1 вихідний кадр *input_frame*. В цій функції виконується метод *cv2.cvtColor(),* який приймає 1 вихідний кадр *input_frame* та відповідний код, який відповідає за перетворення(в даному випадку – це постійні змінні бібліотеки OpenCV, як от *BGR2YUV).*

Наступним є створення екземпляру класу *cv2.VideoCapture(),* параметром якого є шлях до ВЗ та ВК. Далі, для даного екземпляру класу існує один з методів – *cap.isopened(),* який допомагає знайти даний відео-файл за вказаним шляхом.

У випадку, коли в файлі є якась інформація для обробки, цей метод повертає значення «true», якщо виникає помилка або відео закінчується – «false». Попередня операція виконується безпосередньо у циклі *while,* в якому виконується ще один основний метод – *cap.read(зчитування),* при якому відбується зчитування і який повертає 2 вихідні параметри, це *ret(чи цілісним є кадр, з яким працюють)* та *frame.* Далі за допомогою функції *color-space_change()* відповідно перетворюємо ВЗ та ВК у кольоровий простір за варіантом.

Завершальною є функція *cv2.imshow()* бібліотеки OpenCV, яка виконує відображення ВЗ та ВК. Ця функція приймає 2 основні параметри: *'frame'* - титульна інформація даного вікна, в якому буде відображатися кадр та *frame* – сам кадр для відображення. Та не забуваємо зробити 2 функції, одну для кадру «до» та одну для кадру «після» кольорового перетворення.

Після цього додається додатковий цикл з методом *cv2.waitKey(),* який очікує натискання якої-небудь клавіши. Якщо цей метод не додати, то інший метод – *cv2.imshow()* буде просто видалятися. Далі за допомогою методу *release* потрібно звільнити даний відкритий файл з ВЗ. І завершити всі операції з даним кодом необхідно останнім методом *cv2.destroyAllWindows()*, який закриває усі відкриті вікна для демонстрації.

Лістинг коду для зчитування відео з файлу та веб-камери(«до» та «після» кольорового простору «YUV») представлено у додатку А

Екранні форми перевірки представлені у додатку Б

5 ОПЕРАЦІЇ З ВІДЕО-ЗОБРАЖЕННЯМИ

5.1 Можливості бібліотеки OpenCV для виконання операцій

Детектування кутових точок є одним зі способів реалізації комп'ютерного зору. З допомогою цих алгоритмів у кадрі визначають характерні опорні точки для нерухомих об'єктів, які надалі можна використовувати при розрахунку характеристик руху спостережуваних об'єктів. Існує декілька типів таких алгоритмів.

5.2 Метод *cv2.goodFeaturesToTrack()* для *детектування кутів Ши-Томасі з різними параметрами*

Детектор Shi-Tomasi заснований на детекторі Харріса. Різниця закладається в тому, що в детекторі Shi-Tomasi міра відгуку розраховується наступним чином: $R = \min(\lambda 1, \lambda 2)$ Якщо це значення вище певного порогу, то точка вважається кутом і відповідно особливою.

OpenCV має функцію *cv2.goodFeaturesToTrack()* . Він знаходить N найсильніших кутів на зображенні за допомогою методу Ши-Томасі. Як зазвичай, зображення повинно бути в відтінках сірого. Потім ви вказуєте кількість кутів, які ви хочете знайти. Потім ви вказуєте рівень якості, який є значенням від 0 до 1 і позначає мінімальну якість кута, нижче якого всі відхиляються. Потім ми надаємо мінімальну евклідову відстань між виявленими кутами.

Маючи цю інформацію, функція знаходить кути на зображенні. Всі кути нижче за рівень якості відхиляються. Потім він сортує кути, що залишилися за якістю в порядку спадання. Потім функція бере перший найсильніший кут, відкидає всі найближчі кути в діапазоні мінімальної відстані та повертає N найсильніших кутів.

5.3 Реалізація на Python й аналіз результатів

Реалізація на Python починається з підключення бібліотек OpenCV та Numpy, далі створення функції *corner_detector()*, параметрами якої є числові значення *max_corners(максимальний кут), quality_level(рівень якості)* та

min_dist(мінімальна дистанція). В цій функції задаємо у матричному вигляді розміри вихідного зображення(по стовбцях та колонках),за допомогою методу *float32* бібліотеки Numpy задаємо значення, щодо точного розташування зображення відносно країв та за допомогою функції *cv2.getAffineTransform()* бібліотеки OpenCV приймає вхідні дані три пари відповідних точок і виводить 2×3 матрицю афінного перетворення M. В самому кінці основної функції створюємо функцію *cv2.warpAffine(),* яка переміщує зображення вліво та вправо, вниз та вгору, повертаючи готовий результат *img_output.* Наступним є створення екземпляру класу *cv2.VideoCapture(),* параметром якого є шлях до ВЗ та ВК. Далі, для даного екземпляру класу існує один з методів - *cap.isopened(),* який допомагає знайти даний відео-файл за вказаним шляхом.

У випадку, коли в файлі є якась інформація для обробки, цей метод повертає значення «true», якщо виникає помилка або відео закінчується – «false». Попередня операція виконується безпосередньо у циклі *while,* в якому виконується ще один основний метод – *cap.read(зчитування),* при якому відбується зчитування і який повертає 2 вихідні параметри, це *ret(чи цілісним є кадр, з яким працюють)* та *frame*. Далі за допомогою функції *corner_detector()* відповідно геометрично перетворюємо ВЗ та ВК за варіантом.

Завершальною є функція *cv2.imshow()* бібліотеки OpenCV, яка виконує відображення ВЗ та ВК. Ця функція приймає 2 основні параметри: *'frame'* - титульна інформація даного вікна, в якому буде відображатися кадр та *frame* – сам кадр для відображення. Та не забуваємо зробити 2 функції, одну для кадру «до» та одну для кадру «після» кольорового перетворення.

Після цього додається додатковий цикл з методом *cv2.waitKey(),* який очікує натискання якої-небудь клавіши. Якщо цей метод не додати, то інший метод – *cv2.imshow()* буде просто видалятися. Далі за допомогою методу *release* потрібно звільнити даний відкритий файл з ВЗ. І завершити всі операції з даним кодом необхідно останнім методом *cv2.destroyAllWindows()*, який закриває усі відкриті вікна для демонстрації.

Лістинг коду для зчитування відео з файлу(«до» та «після» операції із зображеннями «детектування кутів Ши-Томасі з різними параметрами») представлено у додатку А

Екранні форми перевірки представлені у додатку Б

6 ФІЛЬТРАЦІЯ ВІДЕО-ЗОБРАЖЕНЬ

6.1 Можливості бібліотеки OpenCV з фільтрації

Фільтрація з перетворенням на рельєфне зображення базується на поданні фрагментів зображення з рівномірною яскравістю пікселями сірого кольору, а областей зі значними перепадами яскравості пікселями білого кольору. Це створює ефект тиснення й об'ємності зображення. Ступінь тиснення можна змінити з допомогою різних видів фільтрувальних масок (kernel_emboss). Зазначимо також, що перед процедурою фільтрації кольорове зображення необхідно перетворити на півтонове з допомогою функції cv2.COLOR_BGR2GRAY.

6.2 Метод *cv2.GaussianBlur()* для *фільтра Гауса*

Фільтрація Гауса є фільтрацією згладжувального типу, але, на відміну від прямокутної фільтрації, розмиття буде більш рівномірно розподілятися від центральної точки до країв. Фільтр Гауса є високочастотним фільтром, а отже, його вигідно застосовувати для підвищення різкості зображення.

6.3 Реалізація на Python й аналіз результатів

Реалізація на Python починається з підключення бібліотек OpenCV та Numpy,далі створення функції *gaussian_filtration(),* параметром якої є *kernel_size(розмір ядра)* та повертає більш розмите ВЗ/ВК, але яке буде виглядати природніше, за рахунок добре оброблених країв. Наступним є створення екземпляру класу *cv2.VideoCapture(),* параметром якого є шлях до ВЗ та ВК. Далі, для даного екземпляру класу існує один з методів - *cap.isopened(),* який допомагає знайти даний відео-файл за вказаним шляхом.

У випадку, коли в файлі є якась інформація для обробки, цей метод повертає значення «true», якщо виникає помилка або відео закінчується – «false». Попередня операція виконується безпосередньо у циклі *while,* в якому виконується ще один основний метод – *cap.read(зчитування),* при якому відбується зчитування і який повертає 2 вихідні параметри, це *ret(чи цілісним є*

кадр, з яким працюють) та *frame*. Далі за допомогою функції *gaussian_filtration()* відповідно перетворюємо ВЗ та ВК в геометричне перетворення за варіантом.

Завершальною є функція *cv2.imshow()* бібліотеки OpenCV, яка виконує відображення ВЗ та ВК. Ця функція приймає 2 основні параметри: *'frame'* - титульна інформація даного вікна, в якому буде відображатися кадр та *frame* – сам кадр для відображення. Та не забуваємо зробити 2 функції, одну для кадру «до» та одну для кадру «після» кольорового перетворення.

Після цього додається додатковий цикл з методом *cv2.waitKey(),* який очікує натискання якої-небудь клавіши. Якщо цей метод не додати, то інший метод – *cv2.imshow()* буде просто видалятися. Далі за допомогою методу *release* потрібно звільнити даний відкритий файл з ВЗ. І завершити всі операції з даним кодом необхідно останнім методом *cv2.destroyAllWindows()*, який закриває усі відкриті вікна для демонстрації.

Лістинг коду для зчитування відео з файлу(«до» та «після» фільтрації зображень «Гаусів фільтр з різними масками») представлено у додатку А

Екранні форми перевірки представлені у додатку Б

7 РЕАЛІЗАЦІЯ КЛАСУ ДЛЯ ОБРОБКИ ВІДЕОДАНИХ

7.1 Поняття класу та його створення на 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.

Порівняно з іншими мовами програмування, механізм класів Python додає класи з мінімумом нового синтаксису та семантики. Це суміш механізмів класів, знайдених у C++ і Modula-3. Класи Python забезпечують усі стандартні функції об’єктно-орієнтованого програмування: механізм успадкування класів дозволяє створювати кілька базових класів, похідний клас може перевизначати будь-які методи свого базового класу або класів, а метод може викликати метод базового класу з тим самим іменем. . Об’єкти можуть містити довільні обсяги та типи даних. Як і для модулів, класи мають динамічну природу Python: вони створюються під час виконання та можуть бути змінені далі після створення.

7.2 Реалізація класу *VideoProcessing*

Клас VideoProcessing у Python зазвичай використовується для обробки відеофайлів. Використовуючи цей клас, ви можете обробляти відеопотік та виконувати різні операції, такі як зміна розміру відео, перетворення форматів, додавання фільтрів та ефектів та багато іншого. VideoProcessing у Python дозволяє автоматизувати процес обробки відеоданих та створення професійного відеоконтенту. Розробники можуть використовувати різні бібліотеки, такі як OpenCV, MoviePy, ffmpeg тощо, для обробки відео на Python.

1) Клас повинен, в залежності від номеру вибору завдання, за варіантом виконати зчитування ВЗ/ВК, віддзеркалити їх, змінити кольоровий простір, виконати детектування кутів Ши-Томасі та накласти фільтр Гауса.
2) Перший:filename – string – шлях до ВЗ, другий:mode – integer – чисельне значення для представлення поточного режиму роботи,

третій:mode_to_func – string – функції операцій по порядку , четвертий:mode_to_step – string – чисельне значення порядку виконання операцій та п'ятий:stop_flag – string – помилка.

3) Статичний метод - це метод, який не прив'язаний до стану екземпляра чи класу. Для створення статичного методу використовується декоратор статичного методу. Це підказка для тих, хто читає код, який вказує на те, що метод не залежить від стану екземпляра класу. Більшості методів для роботи потрібне посилання на екземпляр, тому як перший аргумент використовується *self*.

Наступний метод – це *start_processing(),* в якому перевіряються підготовка ВК та ВЗ.

Далі йде *state_check()*, в якому приписуються індекси виклику функцій.

Наступний *modify_frame()*, в якому ми кажемо, що хочемо змінити кадри відповідно до номерів їх функцій.

І останньою є саме функція *main(),* яка буде викликати весь клас.

4) Лістинг коду класу з прикладами виконання усіх задач за варіантом представлені у додатку А;
5) Екранні форми перевірки представлені у додатку Б.

ВИСНОВКИ

Системи ТЗ використовують різноманітні методи обробки відео для аналізу та ідентифікації об'єктів. Від класичних алгоритмів до сучасних нейронних мереж, кожен метод забезпечує унікальне рішення для ефективного управління технологічними процесами на основі візуальної інформації. Завдяки постійному розвитку технології системи ТЗ забезпечують точність та швидкість обробки, відкриваючи нові можливості у різних галузях промисловості та науки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основи побудови сучасних мобільних систем технічного зору [Текст]: навч. посіб. (частина 2). / Л. А. Краснов, К. Ю. Дергачов, С. В. Багінський – Х.: Нац. аерокосм. ун-т ім. М. Е. Жуковського «Харьк. авіац. ін-т», 2018. – 92 с.

2. Колендовська М.М. Харківський національний університет радіоелектроніки 61166, Харків, пр. Науки 14, кафедра МІРЕС, т. 70-21-587 email: d_res@nure.ua

3. Вовк С.М., Гнатушенко В.В., Бондаренко М.В. Методи обробки зображень та комп'ютерний зір. Навчальний посібник. – Д.: «ЛІРА», 2016. – 148 с.

4. Електронний ресурс: https://itmaster.biz.ua/programming/vision/opencv-affinetransform.html - Зміщення зображення в OpenCV. - Автор: Сергій Матвієнко

5. Електронний ресурс: https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.html – Кутовий детектор Shi-Tomasi та гарні можливості для відстеження

ДОДАТОК А

Лістинг коду

Лістинг коду для зчитування відео з файлу:

```
    # підключення необхідних бібліотек
    import cv2

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2': "web"}


    def main():
        # метод зчитування даних з відеофайлу (стор. 142 - 145)
        cap = cv2.VideoCapture(sources.get('video1'))
        # 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)
            if cv2.waitKey(25) == ord('q'):
                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 веб-камери:

```
    # підключення необхідних бібліотек
    import cv2

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2': "web"}


    def main():
     # зчитування відно з веб-камери
        cap = cv2.VideoCapture(0)
        # перевірка готовності веб-камери
        while cap.isOpened():
            # Запис фреймів
            ret, frame = cap.read()
            # при виникненні помилки запису
            if not ret:
                print("Помилка запису файлу")
                break
            # обробка поточного фрейму
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
            # відображення результату
            cv2.imshow('frame', gray)
            if cv2.waitKey(1) == ord('q'):
                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 та веб-камери(«до» та «після» геометричного перетворення «дзеркальне відображення»):

```
    # підключення необхідних бібліотек
    import cv2
    import numpy as np

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2':  "web"}

    # функція дзеркального відображення зображення (стор. 135)
    def mirror(image):
        rows, cols = image.shape[:2]
        src_points = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])
        dst_points = np.float32([[cols - 1, 0], [0, 0], [cols - 1, rows - 1]])
        affine_matrix = cv2.getAffineTransform(src_points, dst_points)
        img_output = cv2.warpAffine(image, affine_matrix, (cols, rows))
        return img_output

    def main():
        cap = cv2.VideoCapture(sources.get('video1'))
        # 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

            # Геометричні перетворення зображення (фрейму)
            frame_mirror = mirror(frame)

            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame_mirrored', frame_mirror)
            if cv2.waitKey(25) == ord('q'):
                break
        # Завершуємо запис у кінці роботи
        cap.release()
        cv2.destroyAllWindows()
        # зчитування відно з веб-камери
        cap = cv2.VideoCapture(0)
        # перевірка готовності веб-камери
        while cap.isOpened():
```

```
            # Запис фреймів
            ret, frame = cap.read()
            # при виникненні помилки запису
            if not ret:
                print("Помилка запису файлу")
                break

            # Геометричні перетворення зображення (фрейму)
            frame_mirror = mirror(frame)

            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame_mirrored', frame_mirror)
            if cv2.waitKey(25) == ord('q'):
                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 та веб-камери(«до» та «після» кольорового простору «YUV»):

```
    # підключення необхідних бібліотек
    import cv2

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2':  "web"}
    colorspaces = {'YUV': cv2.COLOR_BGR2YUV}


    def colorspace_change(input_frame, colorspace_index):
        return cv2.cvtColor(input_frame, colorspaces.get(colorspace_index))


    def main():
        # метод зчитування даних з відеофайлу (стор. 142 - 145)
        cap = cv2.VideoCapture(sources.get('video1'))
        # 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)
            frame_YUV = colorspace_change(frame, 'YUV')

            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame YUV', frame_YUV)
            if cv2.waitKey(25) == ord('q'):
                break
```

```
        # Завершуємо запис у кінці роботи
        cap.release()
        cv2.destroyAllWindows()

        # зчитування відно з веб-камери
        cap = cv2.VideoCapture(0)
        # перевірка готовності веб-камери
        while cap.isOpened():
            # Запис фреймів
            ret, frame = cap.read()
            # при виникненні помилки запису
            if not ret:
                print("Помилка запису файлу")
                break

            # Зміна колірного простору зображення (фрейму)
            frame_YUV = colorspace_change(frame, 'YUV')

            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame YUV', frame_YUV)
            if cv2.waitKey(25) == ord('q'):
                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(«до» та «після» операції із зображеннями «детектування кутів Ши-Томасі з різними параметрами»):

```
    # підключення необхідних бібліотек
    import cv2
    import numpy as np

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2':  "web"}




    # функція детектування кутів Ши-Томасі  (стор. 140)
    def corner_detector(image, max_corners=5, quality_level=0.01, min_dist=20):
        new_image = image.copy()
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        corners     =     cv2.goodFeaturesToTrack(gray_image,     max_corners,
quality_level, min_dist)
        corners = np.float32(corners)
        for item in corners:
            x, y = item[0]
            cv2.circle(new_image, (int(x), int(y)), 5, 255, -1)
        return new_image


    def main():
```

```
        # метод зчитування даних з відеофайлу (стор. 142 - 145)
        cap = cv2.VideoCapture(sources.get('video1'))
        # 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)
            if cv2.waitKey(25) == ord('q'):
                break

            # Виконання операції за варіантом
            frame_changed     =     corner_detector(frame,     max_corners=20,
quality_level=0.01, min_dist=50)

            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame_changed', frame_changed)
            if cv2.waitKey(25) == ord('q'):
                break
        # Завершуємо запис у кінці роботи
        cap.release()
        cv2.destroyAllWindows()

    # зчитування відно з веб-камери
        cap = cv2.VideoCapture(0)
        # перевірка готовності веб-камери
        while cap.isOpened():
            # Запис фреймів
            ret, frame = cap.read()
            # при виникненні помилки запису
            if not ret:
                print("Помилка запису файлу")
                break

            # Виконання операції за варіантом
            frame_changed     =     corner_detector(frame,     max_corners=20,
quality_level=0.01, min_dist=50)

            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame_changed', frame_changed)
            if cv2.waitKey(25) == ord('q'):
                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(«до» та «після» фільтрації зображень «Гаусів фільтр з різними масками»):

```
    # підключення необхідних бібліотек
    import cv2
    import numpy as np

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2':  "web"}


    # Фільтр Гауса
    def gaussian_filtration(image, kernel_size=3):
        return cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)



    def main():
        # метод зчитування даних з відеофайлу (стор. 142 - 145)
        cap = cv2.VideoCapture(sources.get('video1'))
        # 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)
            if cv2.waitKey(25) == ord('q'):
                break

            # Виконання операції за варіантом
            frame_changed = gaussian_filtration(frame, kernel_size=11)


            # Відображення результату
            cv2.imshow('frame', frame)
            cv2.imshow('frame_changed', frame_changed)
            if cv2.waitKey(25) == ord('q'):
                break
        # Завершуємо запис у кінці роботи
        cap.release()
        cv2.destroyAllWindows()

    # зчитування відно з веб-камери
        cap = cv2.VideoCapture(0)
        # перевірка готовності веб-камери
        while cap.isOpened():
            # Запис фреймів
            ret, frame = cap.read()
            # при виникненні помилки запису
            if not ret:
                print("Помилка запису файлу")
                break

            # Виконання операції за варіантом
            frame_changed = gaussian_filtration(frame, kernel_size=11)

            # Відображення результату
```

```
            cv2.imshow('frame', frame)
            cv2.imshow('frame_changed', frame_changed)
            if cv2.waitKey(25) == ord('q'):
                break
        # Завершуємо запис у кінці роботи
        cap.release()
        cv2.destroyAllWindows()

    # при запуску як головного файлу
    if __name__ == '__main__':
        main()
```

Лістинг коду класу з прикладами виконання усіх задач за варіантом:

```
    import cv2
    import numpy as np

    sources = {'video1': "C:/Users/User/PycharmProjects/oop/video/bunnies.mp4",
'video2':  0}


    # клас обробки відео-даних
    class VideoProcessing:

        # конструктор
        def __init__(self, filename):
            self.__filename = filename
            self.__mode = 0
            self.__mode_to_func  =  {1:  VideoProcessing.colorspace_change,  2:
VideoProcessing.mirror,
                                     3:    VideoProcessing.corner_detector,    4:
VideoProcessing.gaussian_filtration}
            self.__mode_to_step = {0: "Basic", 1: "Colorspace", 2: "Geometric",
3: "Operation", 4: "Filtration"}
            self.__stop_flag = False

        def change_input(self, new_filename):
            self.__filename = new_filename

        def set_mode(self, mode_key):
            self.__mode = mode_key

        def get_current_mode_name(self):
            return self.__mode_to_step.get(self.__mode)

        @staticmethod
        def colorspace_change(input_frame):
            return cv2.cvtColor(input_frame, cv2.COLOR_BGR2YUV)

        @staticmethod
        def mirror(input_frame):
            rows, cols = input_frame.shape[:2]
            src_points = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])
            dst_points = np.float32([[cols - 1, 0], [0, 0], [cols - 1, rows -
1]])
            affine_matrix = cv2.getAffineTransform(src_points, dst_points)
            img_output  =  cv2.warpAffine(input_frame,  affine_matrix,  (cols,
rows))
```

```
            return img_output

        @staticmethod
        def  corner_detector(input_frame,  max_corners=5,  quality_level=0.01,
min_dist=20):
            new_image = input_frame.copy()
            gray_image = cv2.cvtColor(input_frame, cv2.COLOR_BGR2GRAY)
            corners    =    cv2.goodFeaturesToTrack(gray_image,    max_corners,
quality_level, min_dist)
            corners = np.float32(corners)
            for item in corners:
                x, y = item[0]
                cv2.circle(new_image, (int(x), int(y)), 5, 255, -1)
            return new_image

        @staticmethod
        def gaussian_filtration(input_frame, kernel_size=3):
            return cv2.GaussianBlur(input_frame, (kernel_size, kernel_size), 0)

        # 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("Помилка запису фрейму!")
                    break

                frame_changed = self.__modify_frame(frame)

                # Відображення результату
                cv2.imshow('frame', frame)
                cv2.imshow('frame_changed', frame_changed)
                self.__state_check()
            # Завершуємо запис у кінці роботи
            cap.release()
            cv2.destroyAllWindows()
            # зчитування відео з веб-камери
            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 помилки запису
                if not ret:
                    print("Помилка запису файлу")
                    break

                    # Відображення результату
                    cv2.imshow('frame', frame)
                    cv2.imshow('frame_changed', frame_changed)
                    self.__state_check()
                # Завершуємо запис у кінці роботи
                cap.release()
```

```
            cv2.destroyAllWindows()

    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())


    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():
    video_processing = VideoProcessing(sources.get('video1'))
    video_processing.start_processing()

    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

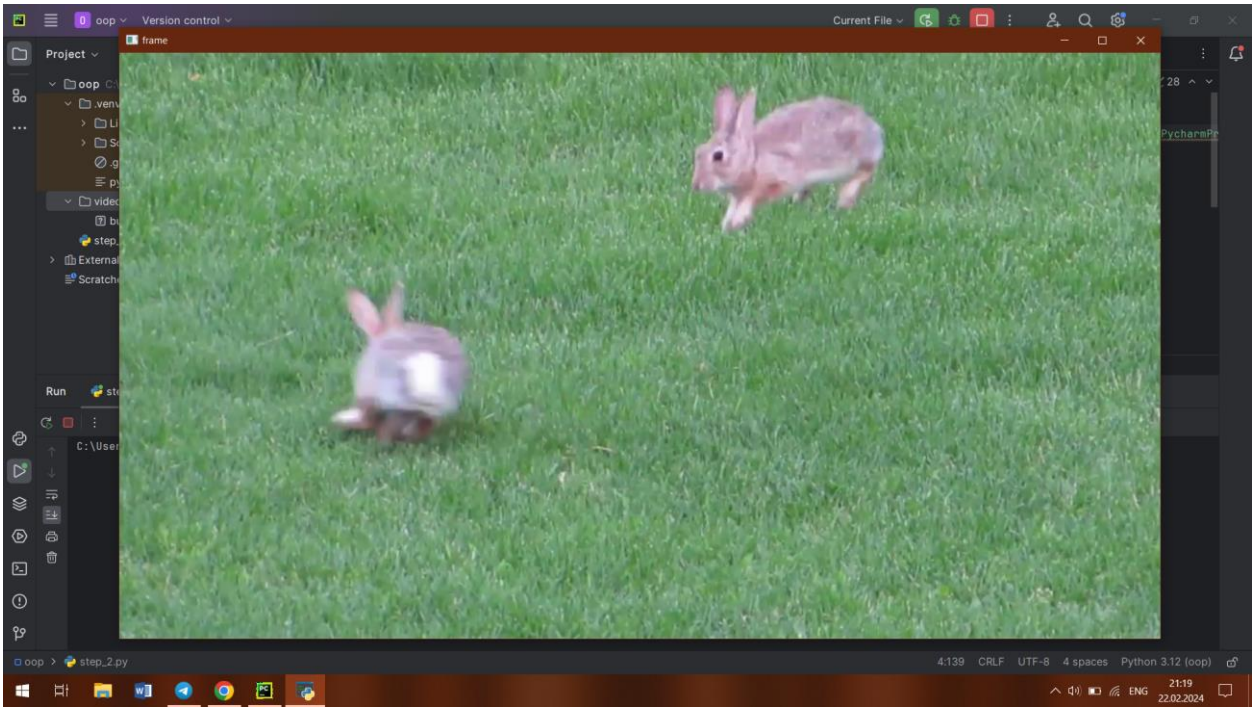

Рисунок Б.1 – Зчитування відео з файлу

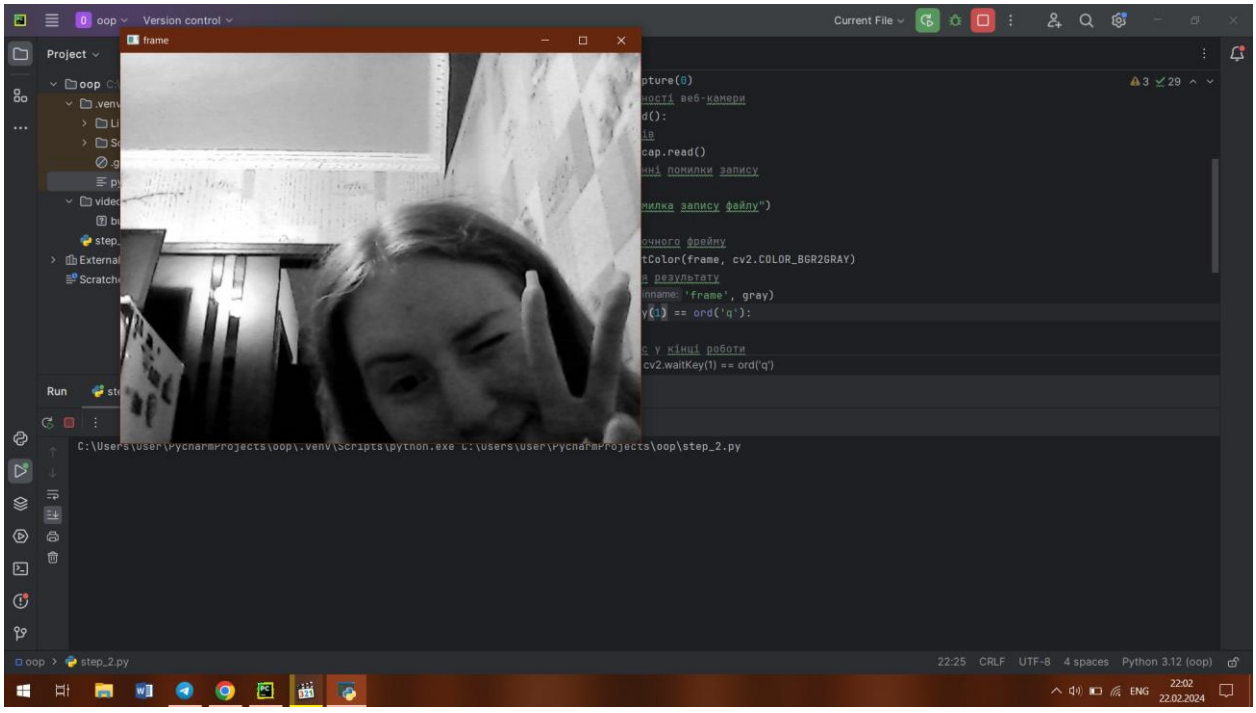

Рисунок Б.2 – Зчитування відео з веб-камери

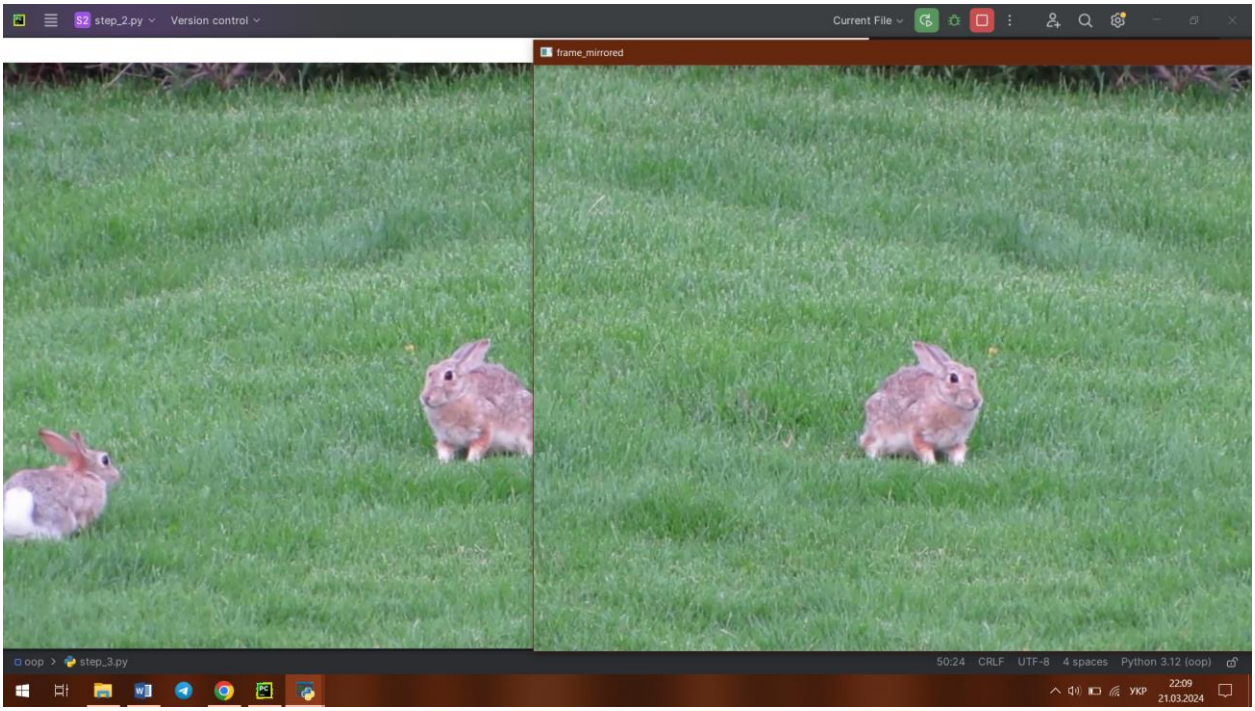

Рисунок Б.3 – Зчитування відео з файлу(«до» та «після» геометричного перетворення «дзеркальне відображення»)

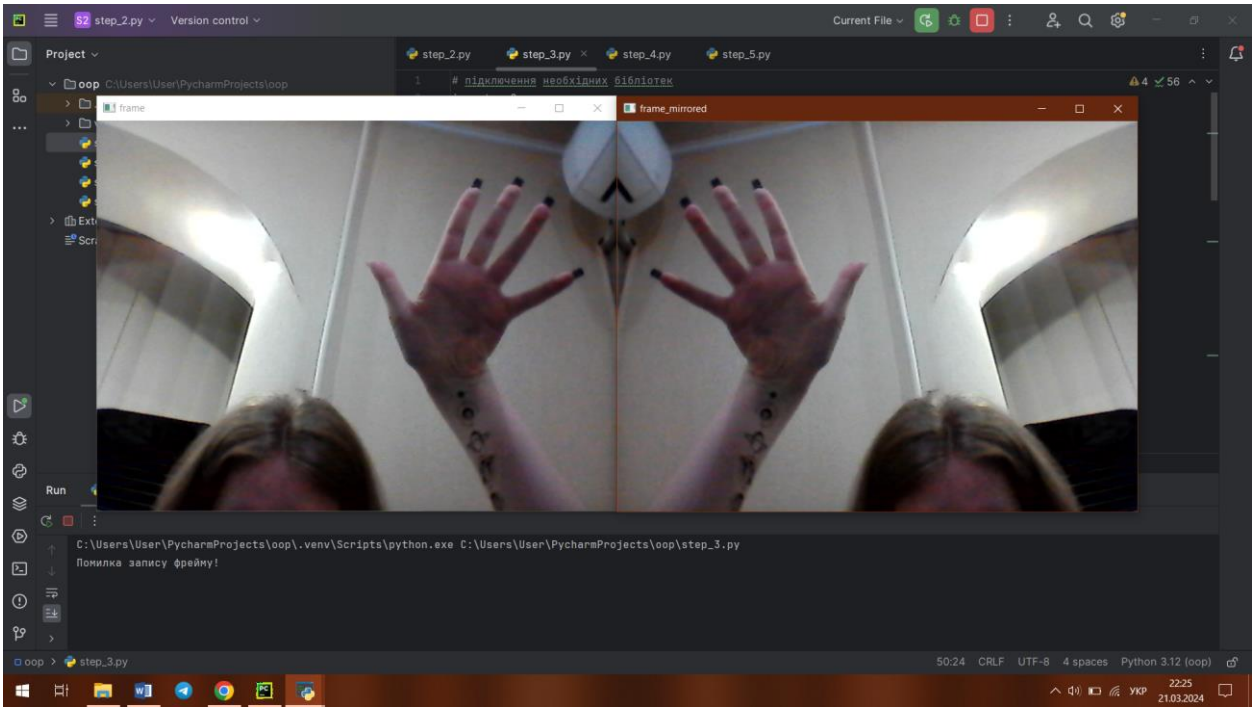

Рисунок Б.4 – Зчитування відео з веб-камери(«до» та «після» геометричного перетворення «дзеркальне відображення»)

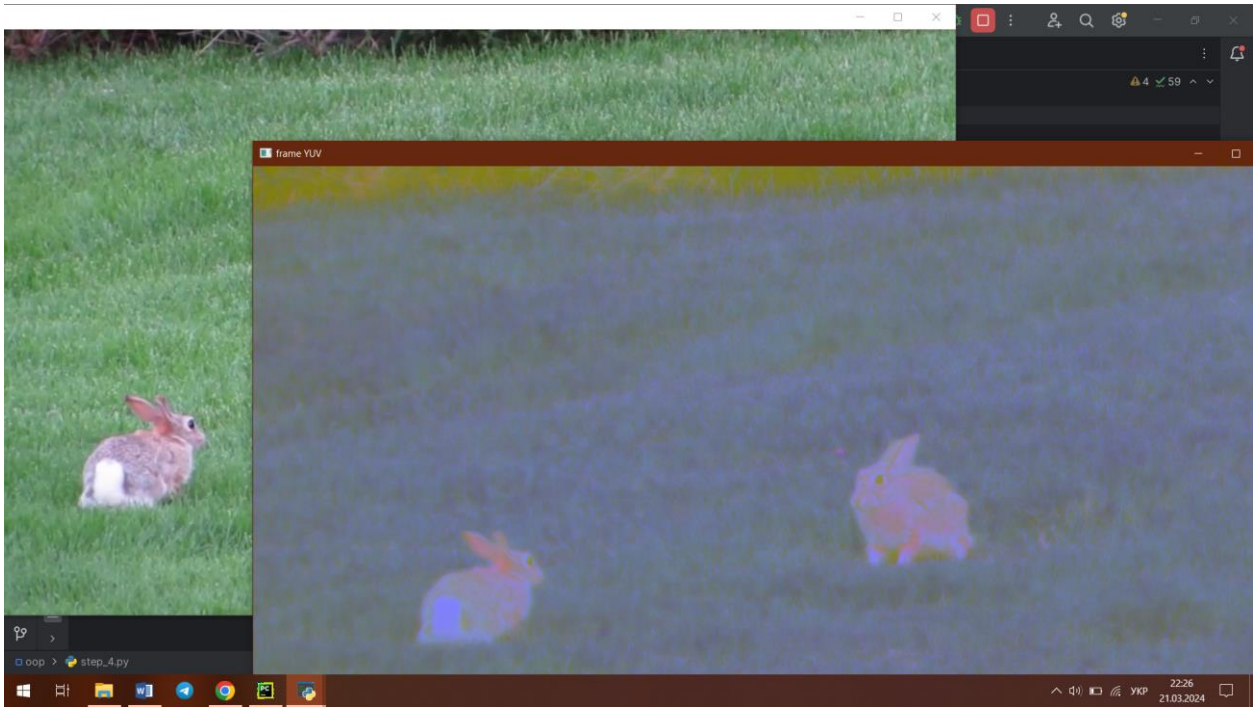

Рисунок Б.5 – Зчитування відео з файлу(«до» та «після» кольорового простору «YUV»)

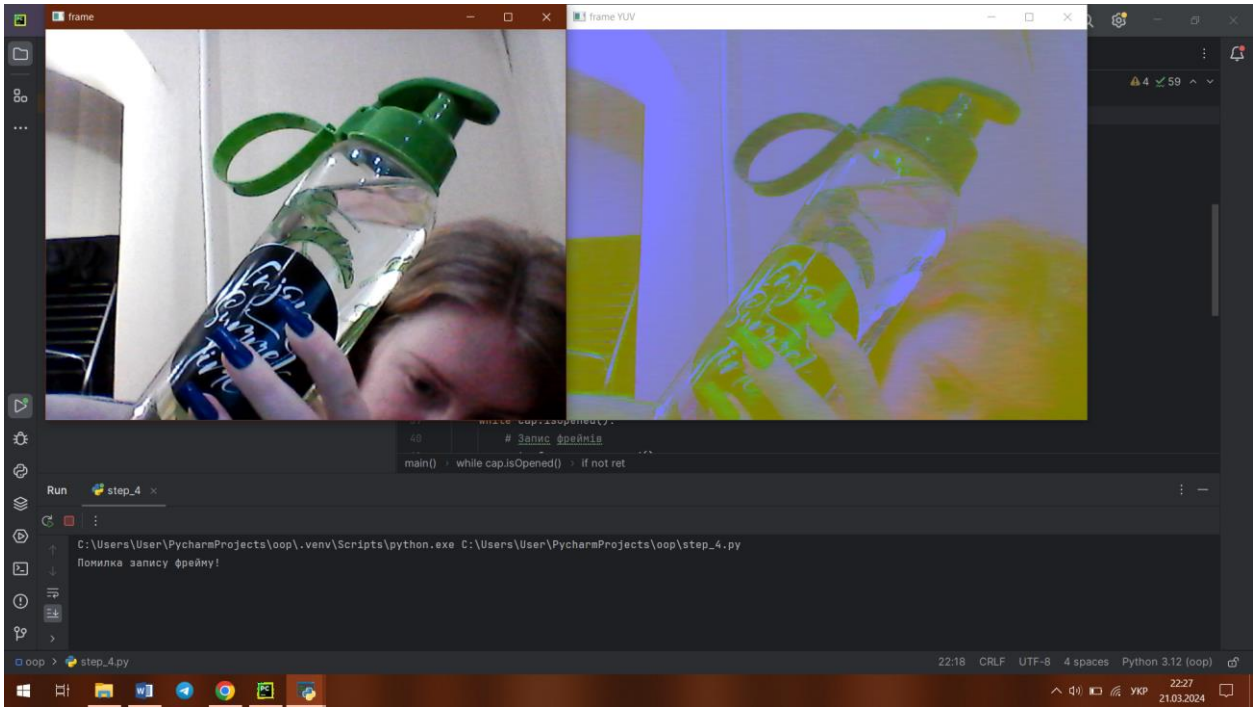

Рисунок Б.6 – Зчитування відео з веб-камери(«до» та «після» кольорового простору «YUV»)

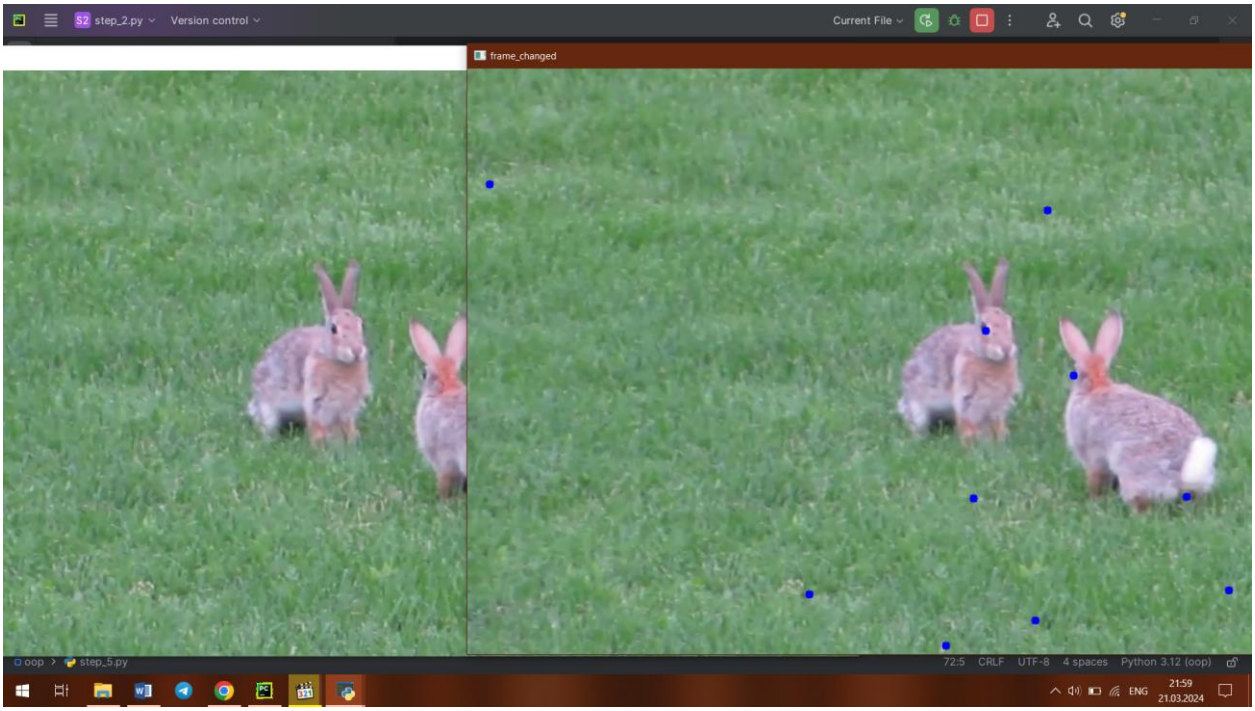

Рисунок Б.7 – Зчитування відео з файлу(«до» та «після» операції із зображеннями «детектування кутів Ши-Томасі з різними параметрами»)

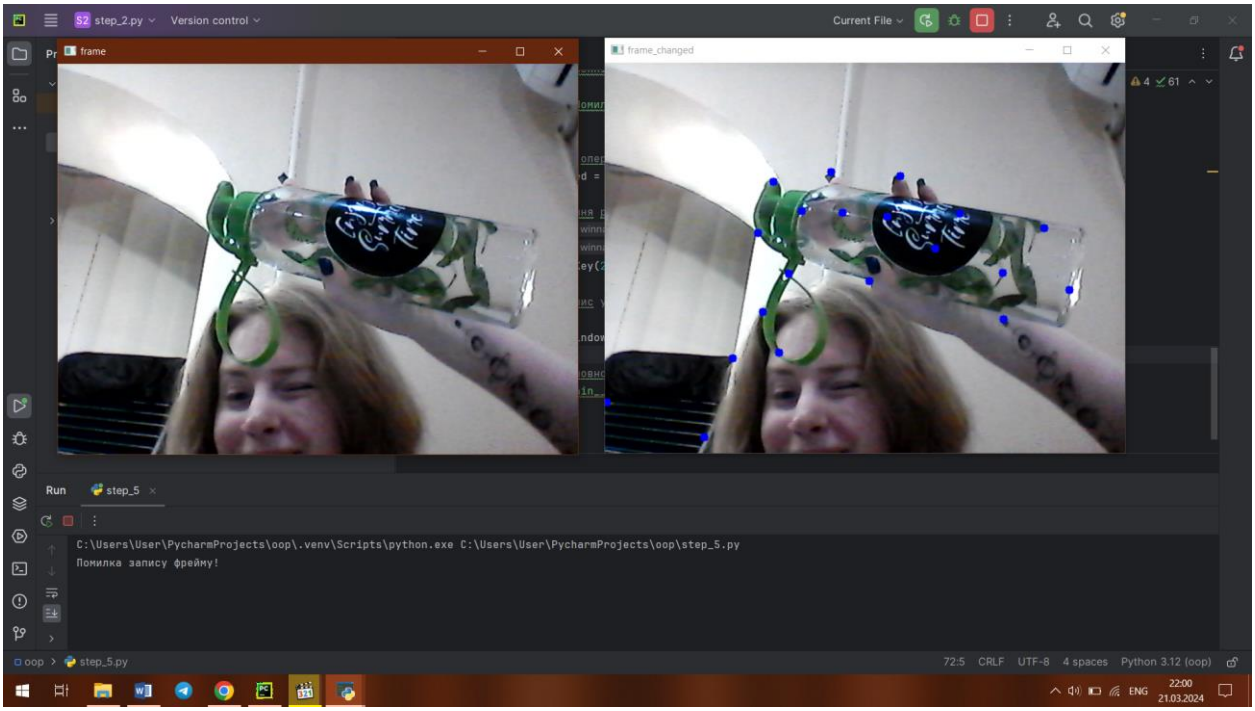

Рисунок Б.8 – Зчитування відео з веб-камери(«до» та «після» операції із зображеннями «детектування кутів Ши-Томасі з різними параметрами»)

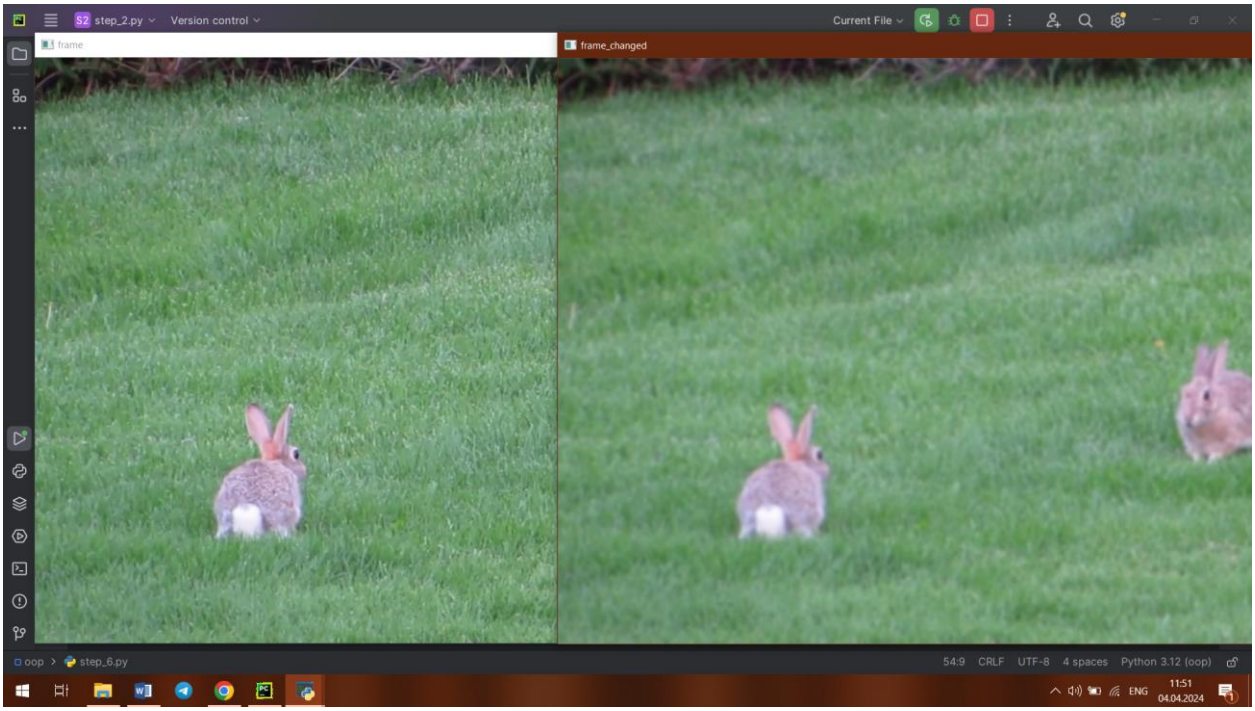

Рисунок Б.9 – Зчитування відео з файлу(«до» та «після» фільтрації зображень «Гаусів фільтр з різними масками»)

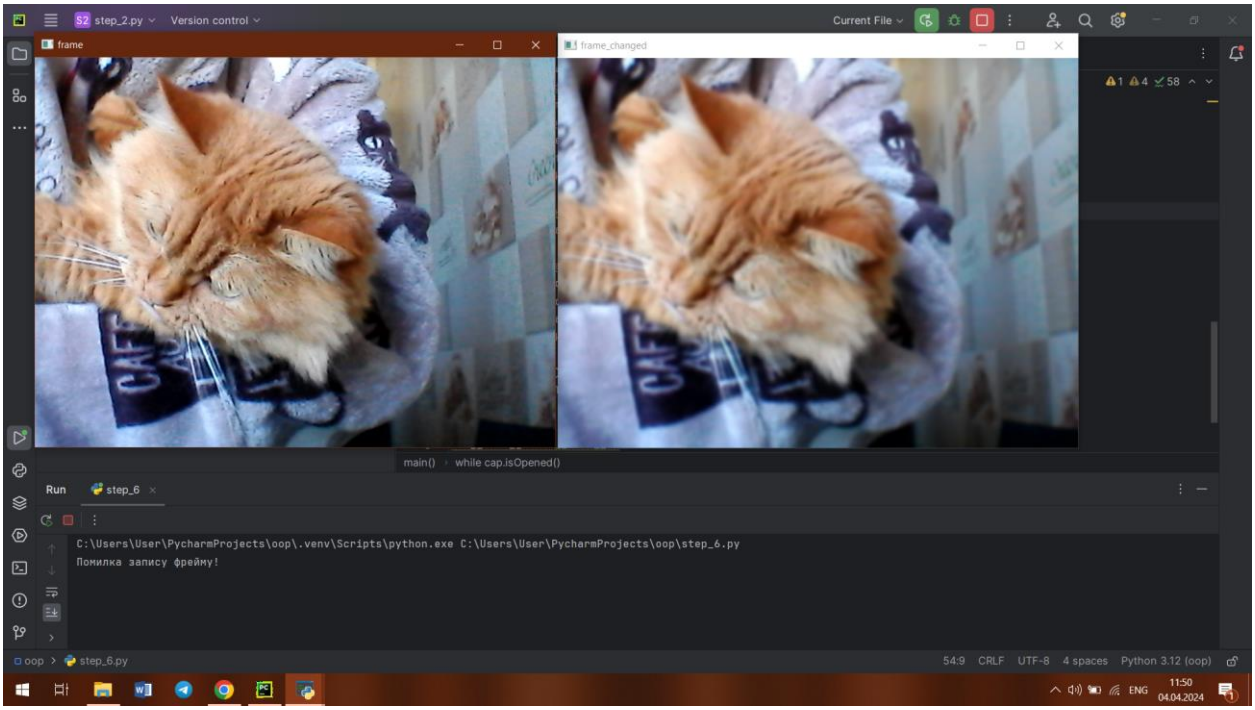

Рисунок Б.10 – Зчитування відео з веб-камери(«до» та «після» фільтрації зображень «Гаусів фільтр з різними масками»)

Рисунок Б.11 – Зчитування відео з файлу(«до» та «після» реалізації класу для оброки відеоданих)

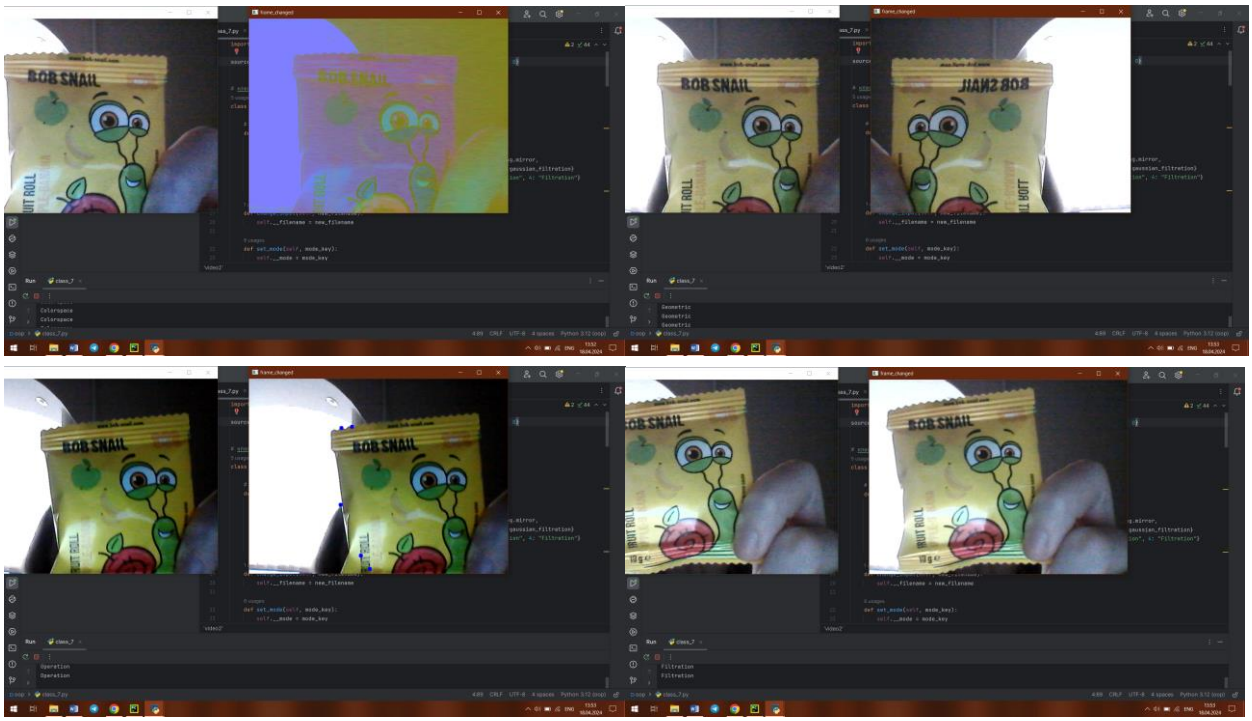

Рисунок Б.12 – Зчитування відео з веб-камери(«до» та «після» реалізації класу для оброки відеоданих)