Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Технології Computer Vision»

на тему

«ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ПОРІВНЯННЯ ЦИФРОВИХ ЗОБРАЖЕНЬ ДЛЯ СТЕЖЕННЯ ЗА ОБ’ЄКТАМИ У ВІДЕОПОТОЦІ»

Виконала:
студентка групи ІС-12
Павлова Софія

Перевірив:
Баран Д. Р.

Київ – 2024

1. Постановка задачі

Мета роботи:

Дослідити принципи та особливості практичного застосування технологій порівняння цифрових зображень для стеження за об’єктами у відеопотоці з використанням спеціалізованих програмних бібліотек.

Завдання ІІ рівня:

Реалізувати дві з наданих груп вимог за власним вибором.

ГРУПА ВИМОГ 1.

Удосконалити скрипт із завдання Лр_5 додавши можливість реалізації порівняння об’єкта ідентифікації (оперативні – високоточні дані ДЗЗ, див. таблицю нижче) з використанням дескриптора зображень.

Вимоги та обмеження:

Дескриптор зображення має стосуватись об’єкта ідентифікації.

Алгоритм та технологія визначення дескриптора зображень – за власним вибором.

Результат порівняння – кількість збігів особливих точок;

Кількість збігів особливих точок дескриптора зображень перевести в ймовірність ідентифікації.

ГРУПА ВИМОГ 3.

Розробити програмний скрипт, що реалізує стеження за об’єктом у цифровому відеопотоці. Зміст відео, об’єкт стеження – обрати самостійно. Метод та технологію стеження обрати такою, що забезпечує стійкість процесу object-tracking для обраних вихідними даними (відео, об’єкт стеження). Вибір обґрунтувати та довести його ефективність.

2. Виконання

2.1. Дескриптор зображень

Доповнимо скрипт програми з попередньої лабораторної роботи, яка ідентифікувала великі поселення, можливістю перевіряти чи не було одне й те саме поселення оброблено двічі. Використаємо для цього дескриптори зображень.

Алгоритм програми

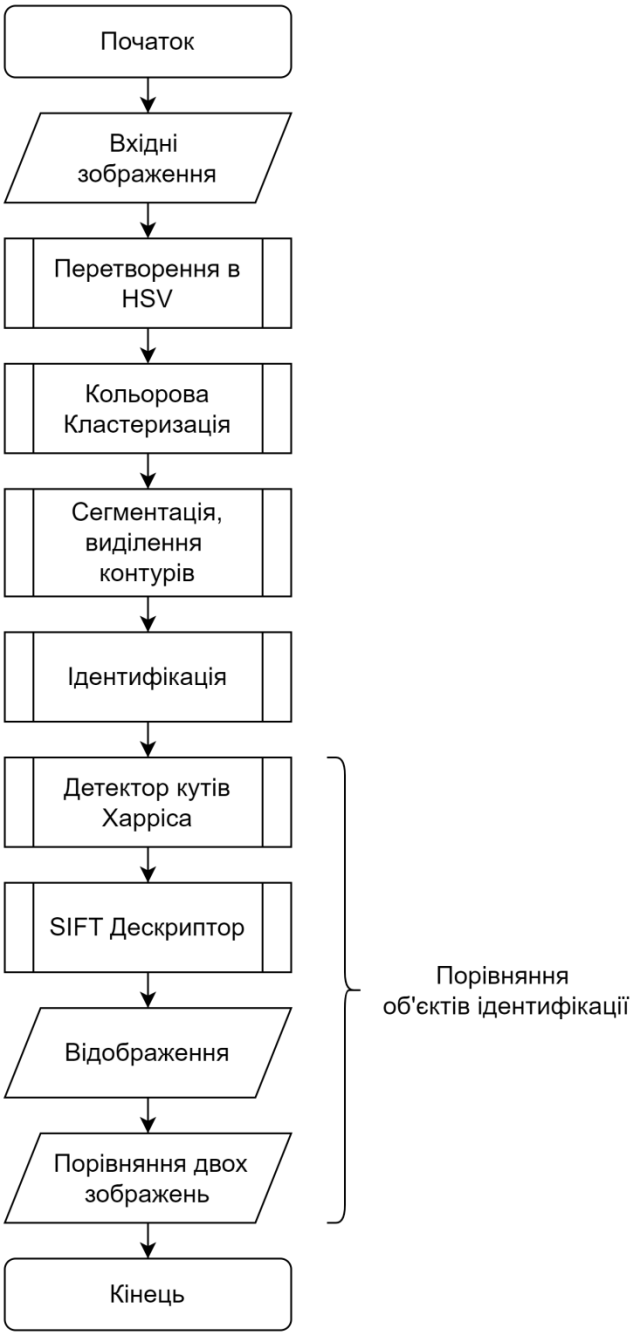


Рисунок 1 – Блок схема програми

Математична модель (Детектор кутів Харріса)

В детекторі Харріса розглядають похідні яскравості зображення для дослідження змін яскравості за множиною напрямків.

Для зображення I встановлено вікно w у центрі (x, y) і його зсув на (u, v). Розмір вікна, як правило, дорівнює 5x5 пікселів, але може залежати від розміру зображення.

Надалі знаходиться різниця в інтенсивності для зміщення (u, v) у всіх напрямках.

Зважена сума квадрата різниць між зрушеним і вихідним вікном (тобто зміна околиці точки (x, y) при зрушенні на (u, v)) дорівнює:

$$E(u, v) = \sum_{x,y} w(x, y) \quad [I(x + u, y + v) - I(x, y)]^2,$$

де $w(x, y)$ – функція вікна (вагова функція Гаусса або одинична функція);

$I(x + u, y + v)$ – зміщена інтенсивність;

$I(x, y)$ – поточна інтенсивність.

Надалі слід максимізувати функцію $E(u, v)$ для виявлення кутів. Застосовуючи розкладання Тейлора до вищенаведеного рівняння та використовуючи деякі математичні спрощення, матимемо остаточне рівняння у вигляді:

$$E(u, v) \approx [u, v] M \begin{bmatrix} u \\ v \end{bmatrix},$$

$$E(u, v) = \sum_{x,y} w(x, y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix},$$

де $I_x I_y$ – похідні яскравості зображення у напрямках x, y.

Надалі вводиться емпіричний спрощувальний вираз:

$$R = \det(M) - k(trace(M))^2,$$

де $det(M) = \lambda_1 \lambda_2, trace(M) = \lambda_1 + \lambda_2, \lambda_1, \lambda_2$ – власні значення M.

Саме величини власних значень визначають, чи є область кутом, ребром чи пласкою.

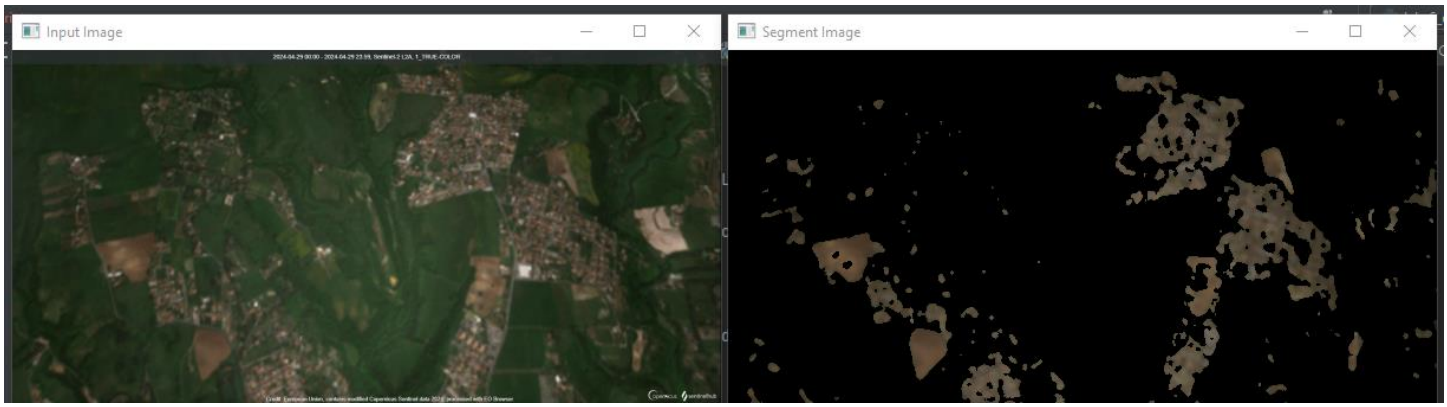
Програмна реалізація

Так, як задача цієї версії скрипта – оцінити, наскільки ідентичними є зображення, виконаємо їх первинну обробку та аналіз їх особливосте по черзі.

У якості першого зображення візьмемо околиці Lazio, що біля Риму, Італія.

Рисунок 2 – Перше зображення

У якості другого візьмемо зображення цього ж міста з 3Д, змінивши трохи кут огляду.



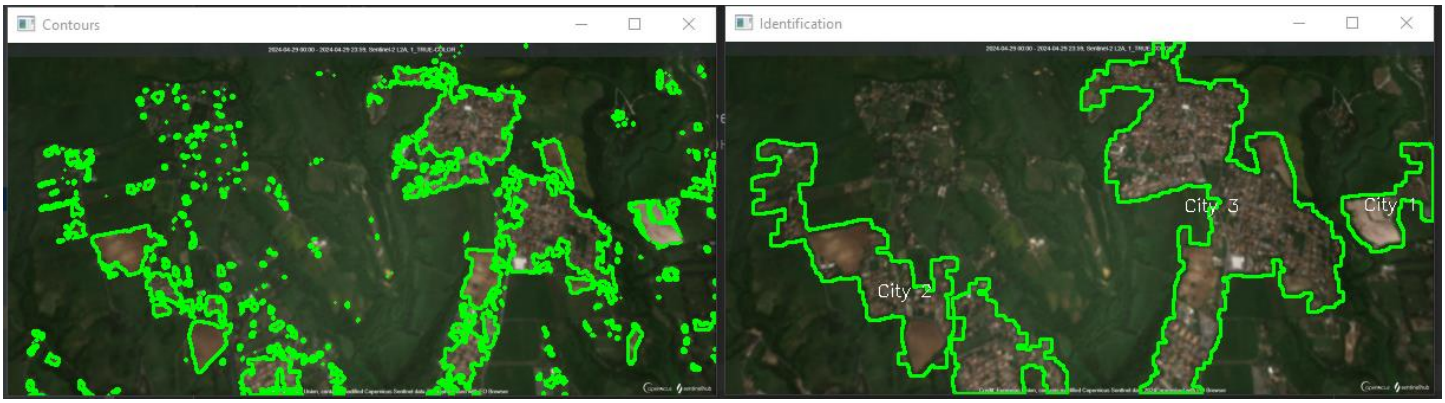


Рисунок 3 – Друге зображення

Детектор кутів Харріса

Для збільшення точності виділення кутів, на вхід для аналізу подамо зображення після кольорової кластеризації.

Математична модель

В детекторі Харріса розглядають похідні яскравості зображення для дослідження змін яскравості за множиною напрямків.

Для зображення I встановлено вікно w у центрі (x, y) і його зсув на (u, v). Розмір вікна, як правило, дорівнює 5x5 пікселів, але може залежати від розміру зображення.

Надалі знаходиться різниця в інтенсивності для зміщення (u, v) у всіх напрямках.

Зважена сума квадрата різниць між зрушеним і вихідним вікном (тобто зміна околиці точки (x, y) при зрушенні на (u, v)) дорівнює:

$$E(u, v) = \sum_{x,y} w(x, y) \quad [I(x + u, y + v) - I(x, y)]^2,$$

де $w(x, y)$ – функція вікна (вагова функція Гаусса або одинична функція);

$I(x + u, y + v)$ – зміщена інтенсивність;

$I(x, y)$ – поточна інтенсивність.

Надалі слід максимізувати функцію $E(u,v)$ для виявлення кутів. Застосовуючи розкладання Тейлора до вищенаведеного рівняння та використовуючи деякі математичні спрощення, матимемо остаточне рівняння у вигляді:

$$E(u,v) \approx [u,v]M\begin{bmatrix}u\\v\end{bmatrix},$$

$$E(u,v) = \sum_{x,y} w(x,y)\ \begin{bmatrix}I_xI_x & I_xI_y\\ I_xI_y & I_yI_y\end{bmatrix},$$

де I_xI_y – похідні яскравості зображення у напрямках x, y.

Надалі вводиться емпіричний спрощувальний вираз:

$$R = \det(M) - k(trace(M))^2,$$

де $det(M) = \lambda_1\lambda_2, trace(M) = \lambda_1 + \lambda_2, \lambda_1, \lambda_2$ – власні значення M.

Саме величини власних значень визначають, чи є область кутом, ребром чи пласкою.

Лістинг коду:

```
# Детектор кутів Харріса
def harris_corner_detector(image):
    img = image.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv2.cornerHarris(gray, 2, 3, 0.04)

    # Розширений результат для розмітки кутів
    dst = cv2.dilate(dst, None)

    # Порогове значення для оптимального значення - може відрізнятися залежно від
зображення
    img[dst > 0.01 * dst.max()] = [0, 0, 255]

    return img
```

Результат:

При порівнянні з оригінальним кадром, бачимо, що алгоритм доволі точно виділив особливості форми поселення.

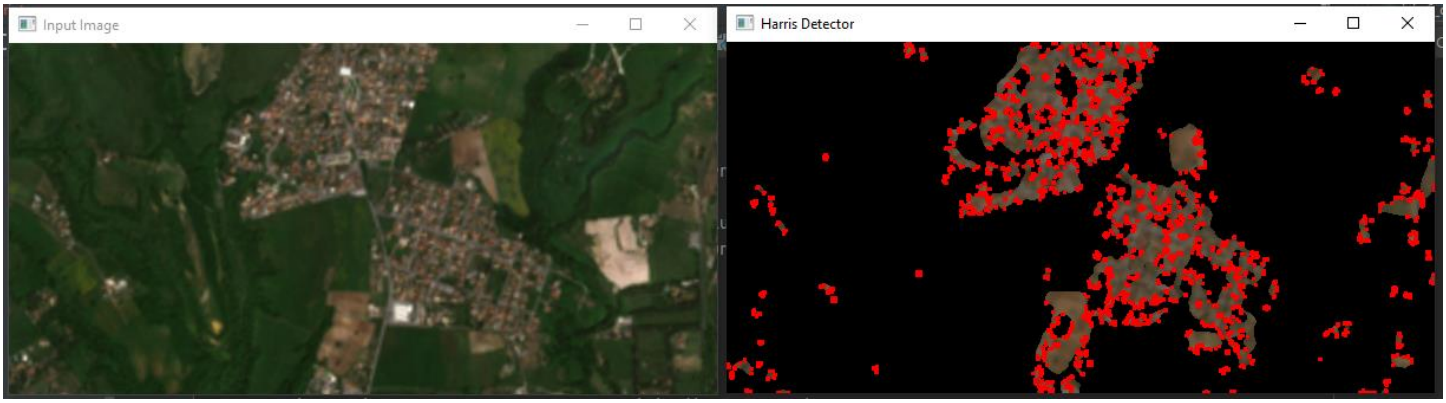


Рисунок 4 – Результат виділення кутів Харрсіа (1)

Рисунок 5 – Результат виділення кутів Харріса (2)

Дескриптор SIFT

Детектор кутів Харріса знаходить особливі точки на різних зображеннях але не дає можливості їх порівняти та встановити відповідність між кутами. Тому необхідно сформувати для кожної особливої точці дескриптор та реалізувати спосіб порівняння дескрипторів.

Математична модель

Дескриптор кутової точки Харріса включає: значення яскравості в оточенні блока зображення (прямокутник з центром в кутовій точці) + нормоване значення взаємної корреляції.

У загальному вигляді корреляція між двома блоками зображення $I_1(x)$ та $I_2(x)$ однакового розміру визначається за виразом:

$$c(I_1, I_2) = \sum_x f(I_1(x), I_2(x)),$$

де f – визначає метод обрахунку корреляції, сума визначається за множиною особливих точок x, нормування здійснюється відносно СКВ яскравості.

Лістинг коду:

```
# Дескриптор SIFT для заданих функцій на основі виявлення кутів Харріса
def sift_descriptors_on_harris(image):

    def harris(img):
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gray_img = np.float32(gray_img)
        dst = cv2.cornerHarris(gray_img, 2, 3, 0.04)
        result_img = img.copy()

        # Порогове значення для оптимального значення - може відрізнятися залежно від
зображення
        # Малює ключові точки кута Харріса на зображенні (RGB [0, 0, 255] -> синій)
        result_img[dst > 0.01 * dst.max()] = [0, 0, 255]
        # dst, більше за порогове значення = ключова точка
        keypoints = np.argwhere(dst > 0.01 * dst.max())
        keypoints = [cv2.KeyPoint(float(x[1]), float(x[0]), 13) for x in keypoints]

        return (keypoints, result_img)

    img = image.copy()
    # Обчислюємо features кута Харріса та перетворюємо їх на ключові точки
    kp, img = harris(img)
    # Обчислюємо дескриптори SIFT з ключових точок Harris Corner
    sift = cv2.SIFT_create()
    sift.compute(img, kp)
    img = cv2.drawKeypoints(img, kp, img)

    return img
```

Результат:

Алгоритм знайшов **на зображенні 1 – 901** ознаку, а **на зображенні 2 – 687** ознак.

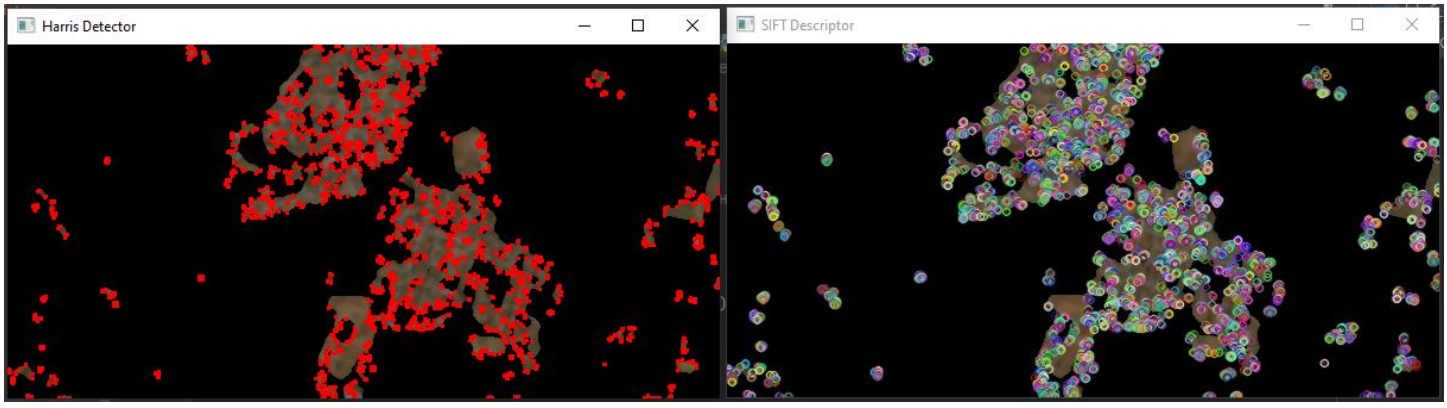


Рисунок 6 – Результат виділення ознак дескриптором SIFT (1)

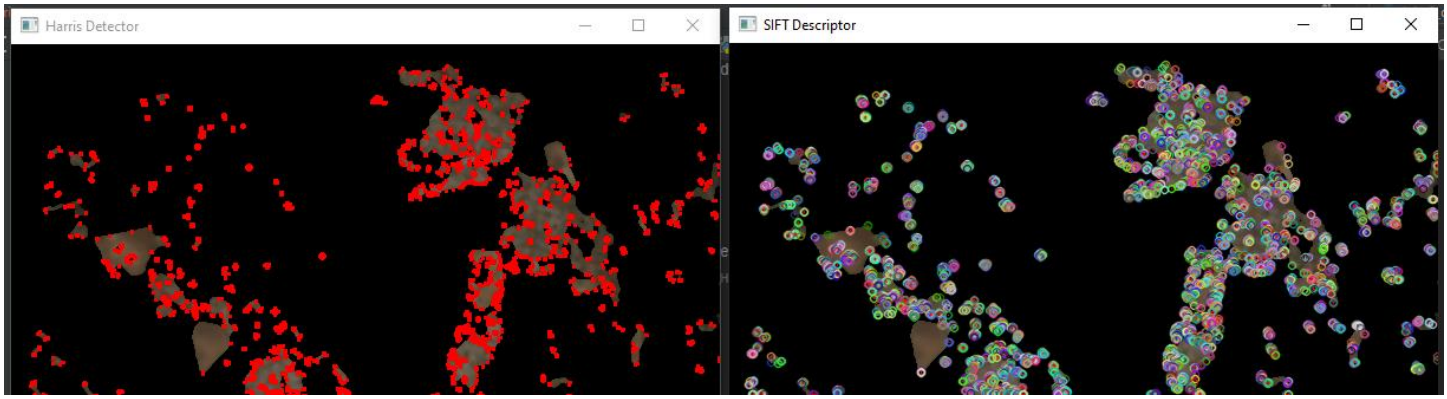


Рисунок 7 – Результат виділення ознак дескриптором SIFT (2)

```
Усього дескрипторів на image 1:     901
Усього дескрипторів на image 2:     687
```

Рисунок 8 – Кількість ідентифікованих дескрипторів

Співставлення ознак

Для порівняння двох зображень, використаємо раніше виявлені ознаки, відобразимо зони їх ідентифікації та визначимо числовий відсоток ідентичності зображень.

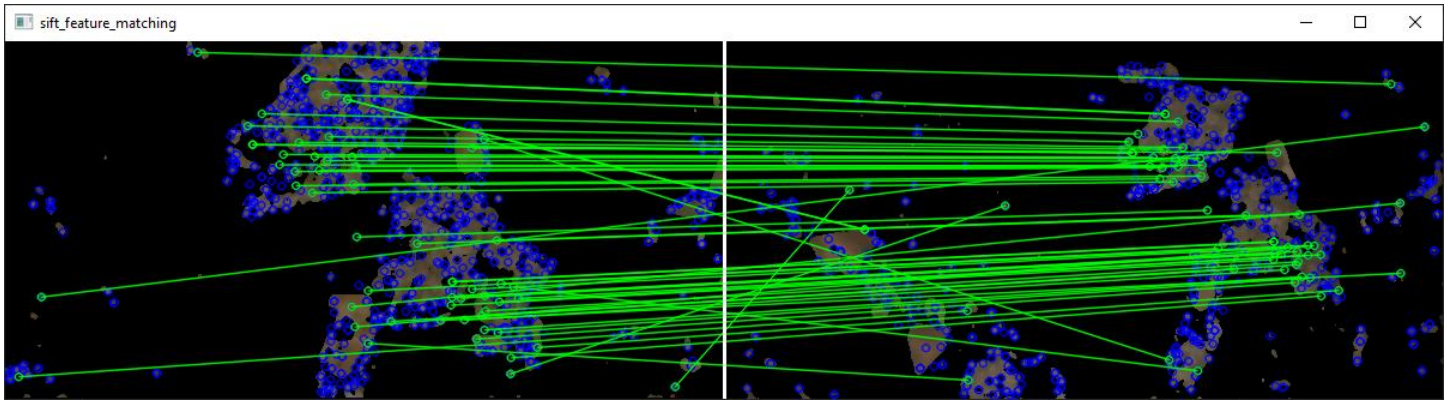
Лістинг коду:

```
# Порівняння двох зображень
def sift_feature_matching(image_1, image_2):
    img1 = image_1.copy()
    img2 = image_2.copy()

    # Запуск детектора SIFT
    sift = cv2.SIFT_create()
    # Знаходження ключових точок та дескрипторів за допомогою SIFT
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    # Параметри FLANN
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)  # or pass empty dictionary
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des1, des2, k=2)

    # Підрахунок кількості ідентифікованих дескрипторів для кожного зображення
    total_descriptors_img1 = len(des1)
    total_descriptors_img2 = len(des2)

    # Підрахунок кількості співпадінь
```

```
    matching_descriptors = 0
    for m, n in matches:
        if m.distance < 0.7 * n.distance:
            matching_descriptors += 1

    # Виведення кількості ідентифікованих дескрипторів для кожного зображення
    print(f'Усього дескрипторів на image 1:\t\t{total_descriptors_img1}')
    print(f'Усього дескрипторів на image 2:\t\t{total_descriptors_img2}')
    print(f'\nКількість співпадінь:\t\t\t{matching_descriptors}')

    # Вибір меншої кількості дескрипторів для розрахунку відсотка ідентичності
    min_total_descriptors = min(total_descriptors_img1, total_descriptors_img2)

    # Розрахунок відсотка ідентичності, з урахуванням маштабування
    similarity_percentage = round(min(100 * (matching_descriptors /
min_total_descriptors), 100), 3)
    print(f'Відсоток ідентичності:\t\t\t{similarity_percentage}%')
    belive_interval = round(min(100 * (matching_descriptors / min_total_descriptors) /
0.15, 100), 3)
    print(f'Відсоток довіри:\t\t\t{belive_interval}%')

    # Створення маски для відображення зіставлених точок на зображенні
    matchesMask = [[0, 0] for i in range(len(matches))]
    for i, (m, n) in enumerate(matches):
        if m.distance < 0.7 * n.distance:
            matchesMask[i] = [1, 0]

    draw_params = dict(matchColor=(0, 255, 0),
                       singlePointColor=(255, 0, 0),
                       matchesMask=matchesMask,
                       flags=cv2.DrawMatchesFlags_DEFAULT)

    img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, matches, None, **draw_params)
    # Розмір зображення для створення білої лінії
    height, width, _ = img3.shape
    # Додавання білої лінії між зображеннями
    img_with_line = cv2.line(img3, (width // 2, 0), (width // 2, height), (255, 255,
255), 2)
    cv2.imshow('sift_feature_matching', img_with_line)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Результат:

У результаті порівняння було знайдено **57 співпадінь**.



```
Кількість співпадінь:             57
```

Рисунок 9 – Результат порівняння двох зображень

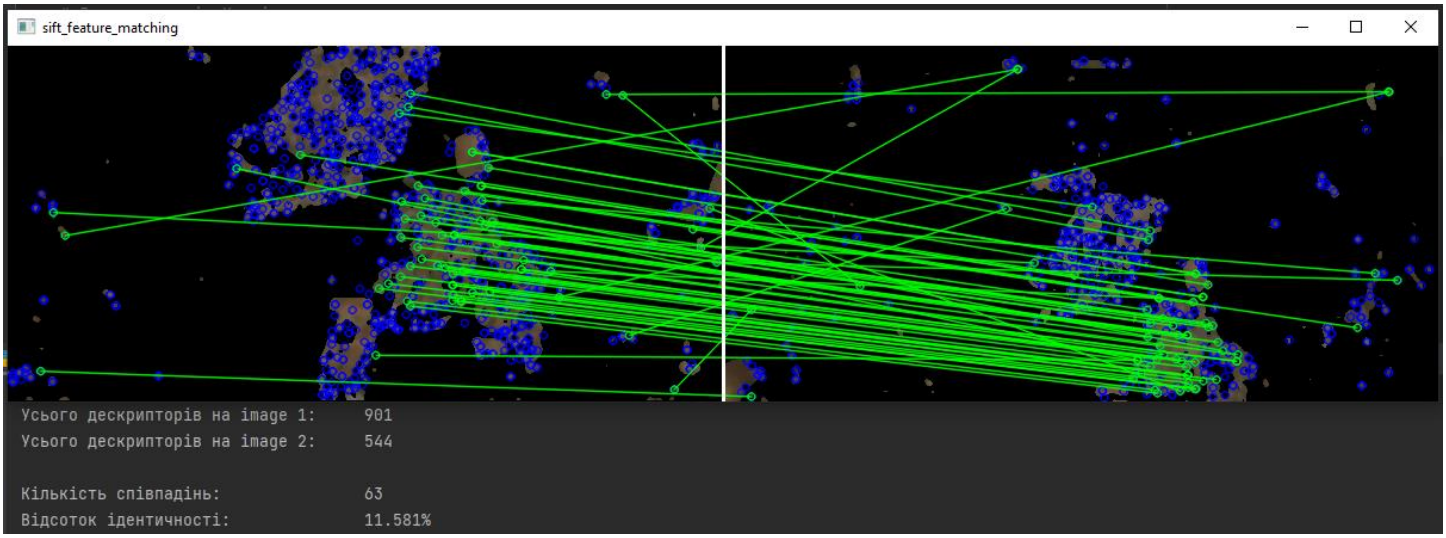
Розрахуємо **відсоток ідентичності об’єктів** (у нашому випадку поселень) знайшовши частку співпадаючих ознак до загальної кількості ознак на одному зображенні (рахувати будемо від того зображення, де ідентифіковано менше ознак).

```
similarity_percentage = round(min(100 * (matching_descriptors / min_total_descriptors), 100), 3)
```

Отримаємо дуже малий відсоток співпадінь – **8.297%**.

```
Відсоток ідентичності:            8.297%
```

Тому, було **проведено серію експериментів** з ще 4-ма фотографіями цього міста з різних ракурсів та масштабів, щоб прослідкувати, на який коефіцієнт можна домножити отримані дані, щоб вони якомога точніше відображали ідентичність розглянутих об’єктів.



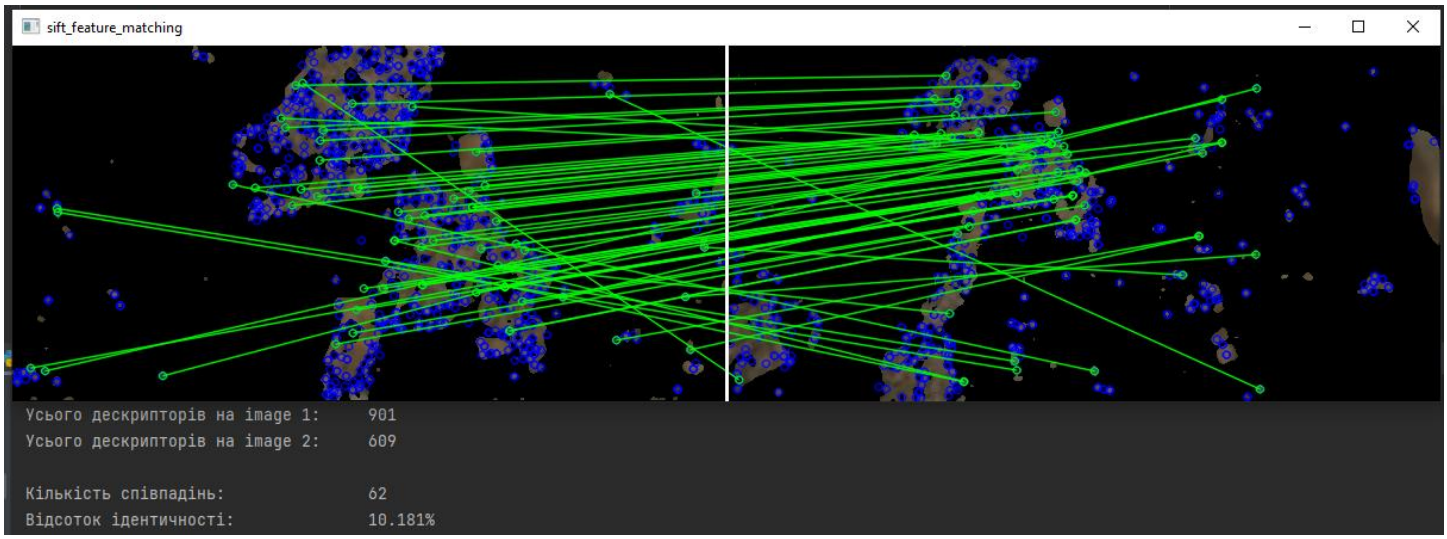


Рисунок 10 – Експерименти з різними ракурсами одного об'єкту

Останнє фото було максимально подібне до оригіналу, це можна навіть бачити за пропорціями, які видно крізь лінії порівняння дескрипторів. Його відсоток ідентичності і було взято за еталонний.

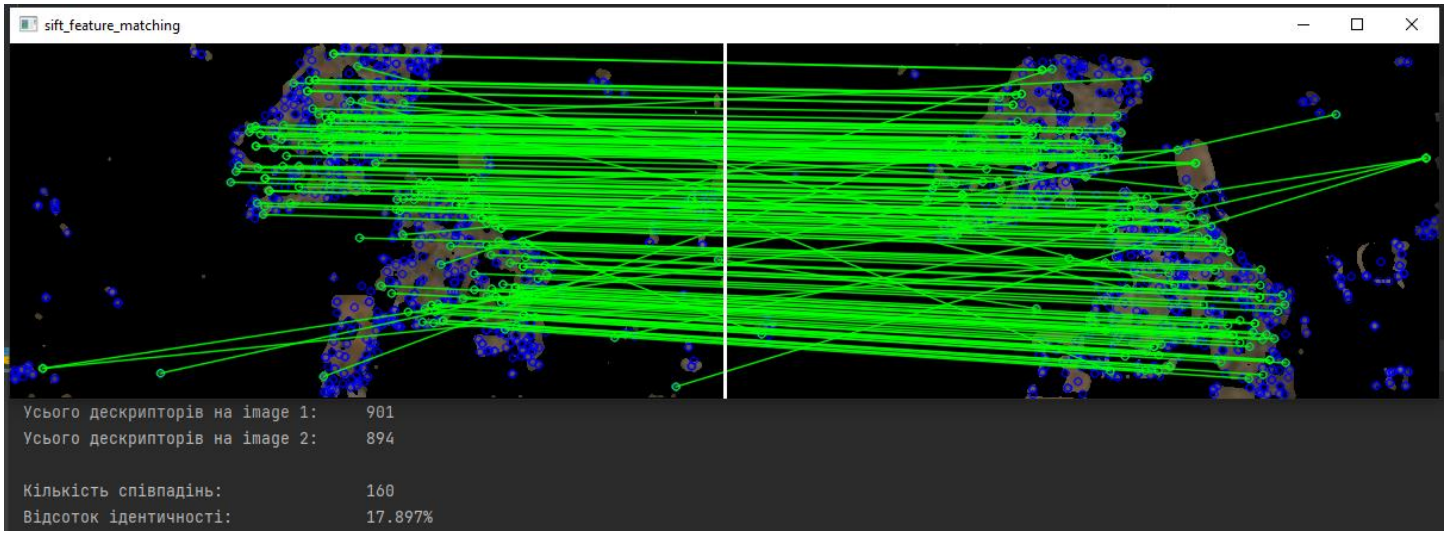


Рисунок 11 – Еталонне значення відсотку

Формулу відсотку було помножено на 0.15, і ось який результат отримано для розглянутих зображень:

Зображення 1 / Зображення 2 – 8.297% (**55.313%**)

Зображення 1 / Зображення 3 – 11.581% (**77.206%**)

Зображення 1 / Зображення 4 – 10.181% (**67.871%**)

Зображення 1 / Зображення 5 – 17.897% (**100%**)

Усі результати > 50% можна вважати ідентичними об’єктами.

Проведемо **контрольний експеримент** з іншим містечком.

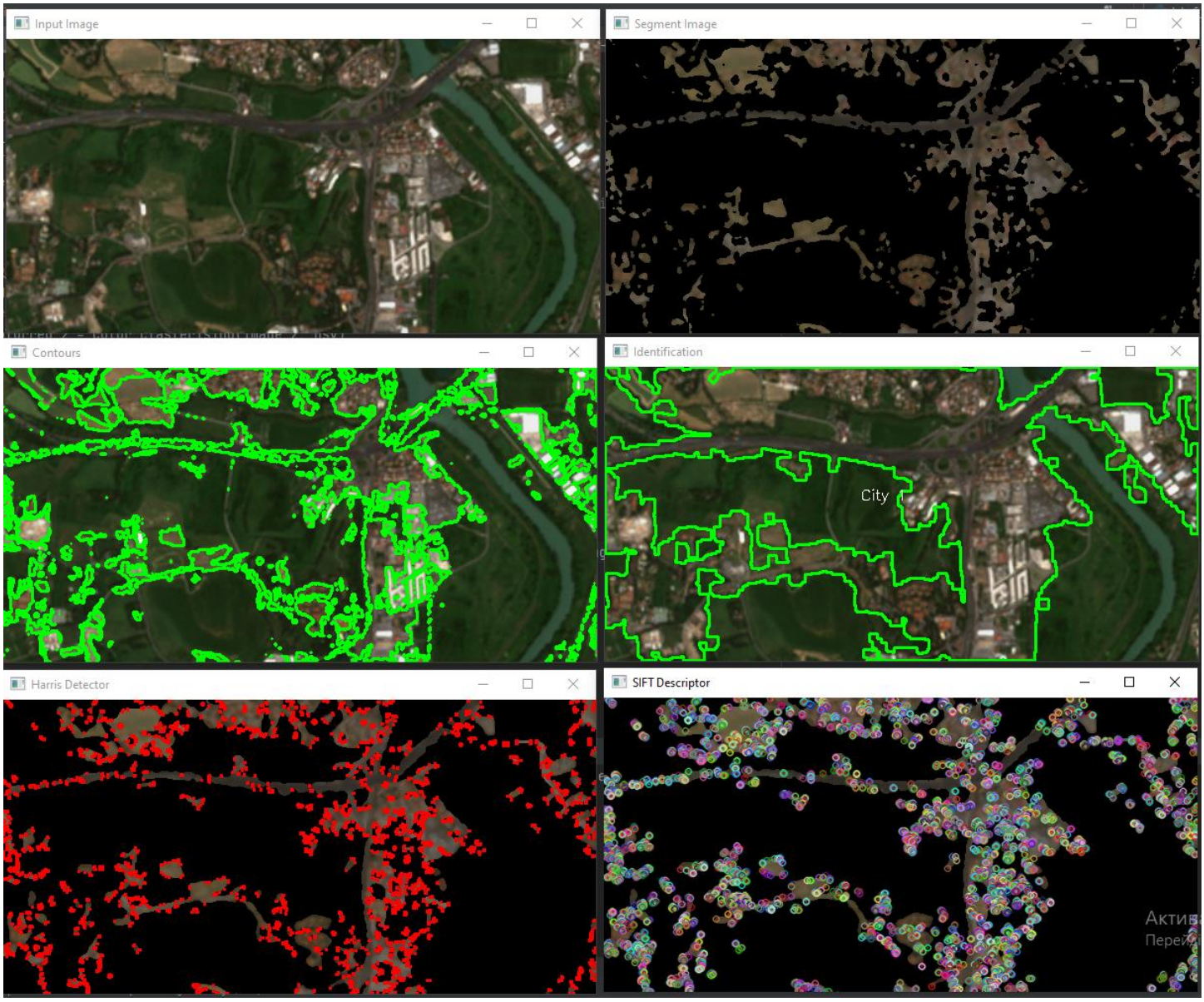


Рисунок 12 – Зображення іншого поселення

У результаті бачимо, що було знайдено найменше співпадінь. Усі вони випадкові і є недоліком алгоритму. Але відсоток довіри **15.538%** підтверджує, що на зображеннях різні об’єкти.

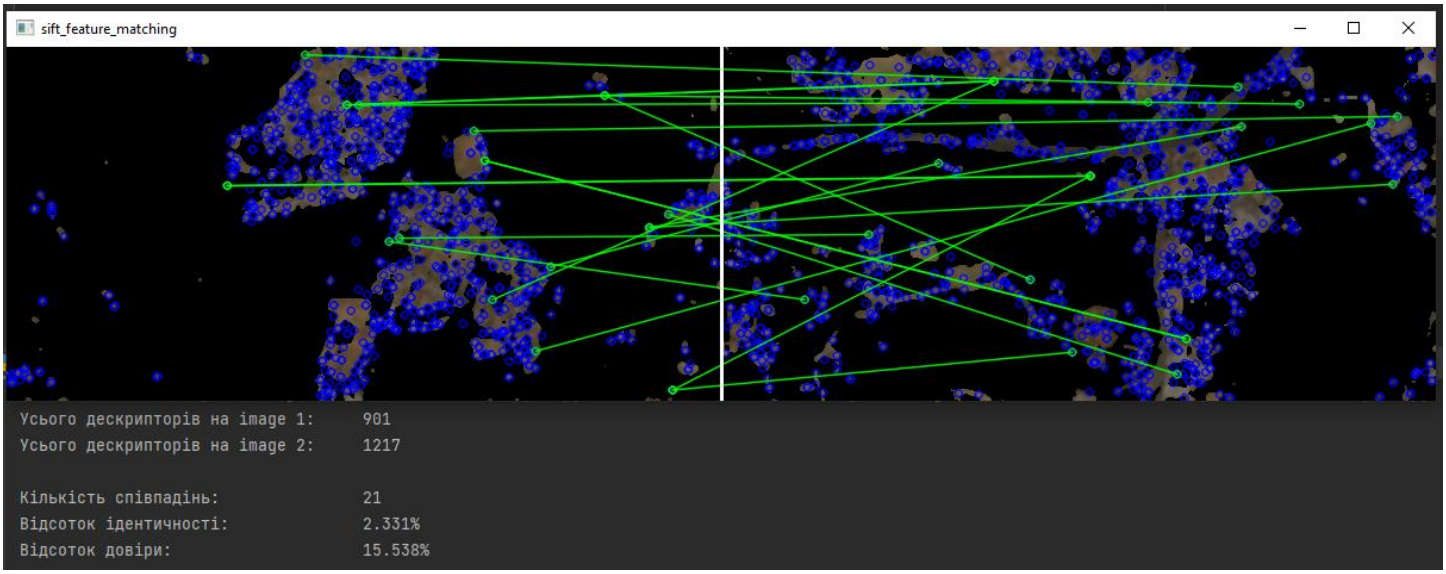


Рисунок 13 – Результат порівняння різних об’єктів

Отже, ми бачимо, що **алгоритм успішно виконує поставлену задачу**.

2.2. Стеження за об’єктом

Поставимо собі задачу **відстежувати людей** у відеопотоці. Підійдемо до розгляду цього питання з точки зору камер для відстеження руху гравців для VR та за стосунків заміни обличчя людини у режимі реального часу на кшталт ReFace.

Для цих технологій важливий трекінг не людини в цілому, а її відкритих частин тіла, а особливо обличчя.

Для цього чудово підійдуть технології **роботи з кольоровими просторами HSV**.

Алгоритм програми

Програма буде зчитувати кадр за кадром відеопотоку, на який буде застосовувати фільтри (маски) різних відтінків шкіри людини.

Рисунок 14 – Блок схема програми

Програмна реалізація

Розробимо програму так, щоб користувач міг використовувати декілька фільтрів для різних відтінків шкіри і порівнювати їх ефективність для кожного відео.

Фільтр для світлої шкіри холодних відтінків

Експериментально було знайдено оптимальне значення відтінку, насиченості та яскравості для блідої шкіри або характерного освітлення.

```
# Фільтр 1
lower_1 = np.array([0, 10, 0])
upper_1 = np.array([25, 170, 255])
```

Фільтр налаштовувався по даному відео.

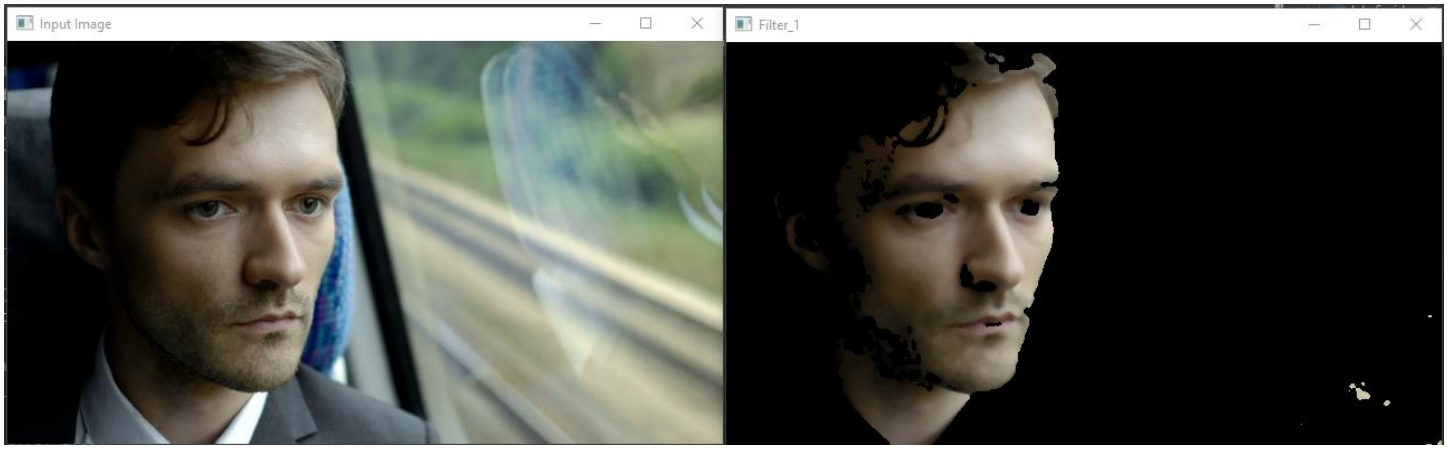


Рисунок 15 – Налаштування фільтру для світлої шкіри

Такий фільтр дуже **чутливий до білого кольору і світлих відтінків інших кольорів**. Тому на багатьох відео може захоплювати зайві об’єкти.

Фільтр для смуглої шкіри теплих відтінків

Експериментально було взято наступні значення для смуглої шкіри і теплого відтінку.

```
# Фільтр 2
lower_2 = np.array([3, 0, 0])
upper_2 = np.array([11, 180, 255])
```

Фільтр налаштовувався по даному відео.

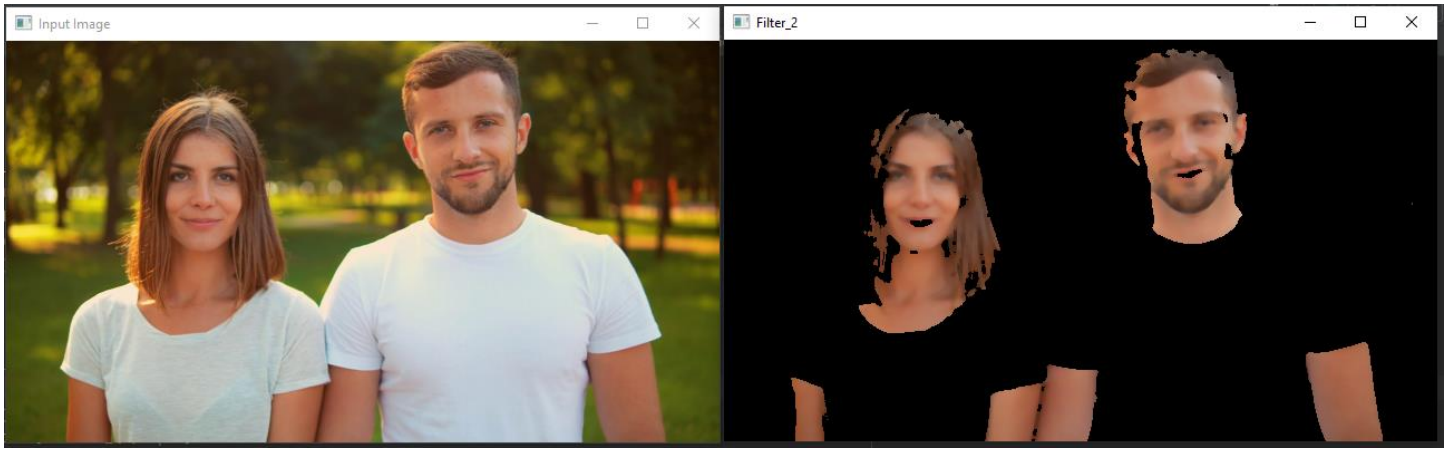


Рисунок 16 – Налаштування фільтру для смуглої шкіри

Такий фільтр **відсікає більшість діапазону, який захоплює попередній фільтр** і **розширюється для охоплення темних кольорів та їх відтінків**. Фільтр часто захоплює волосся та може помилково ідентифікувати зайві темні об’єкти з фону.

Протестуємо створені фільтри на декількох відео і порівняємо результати.

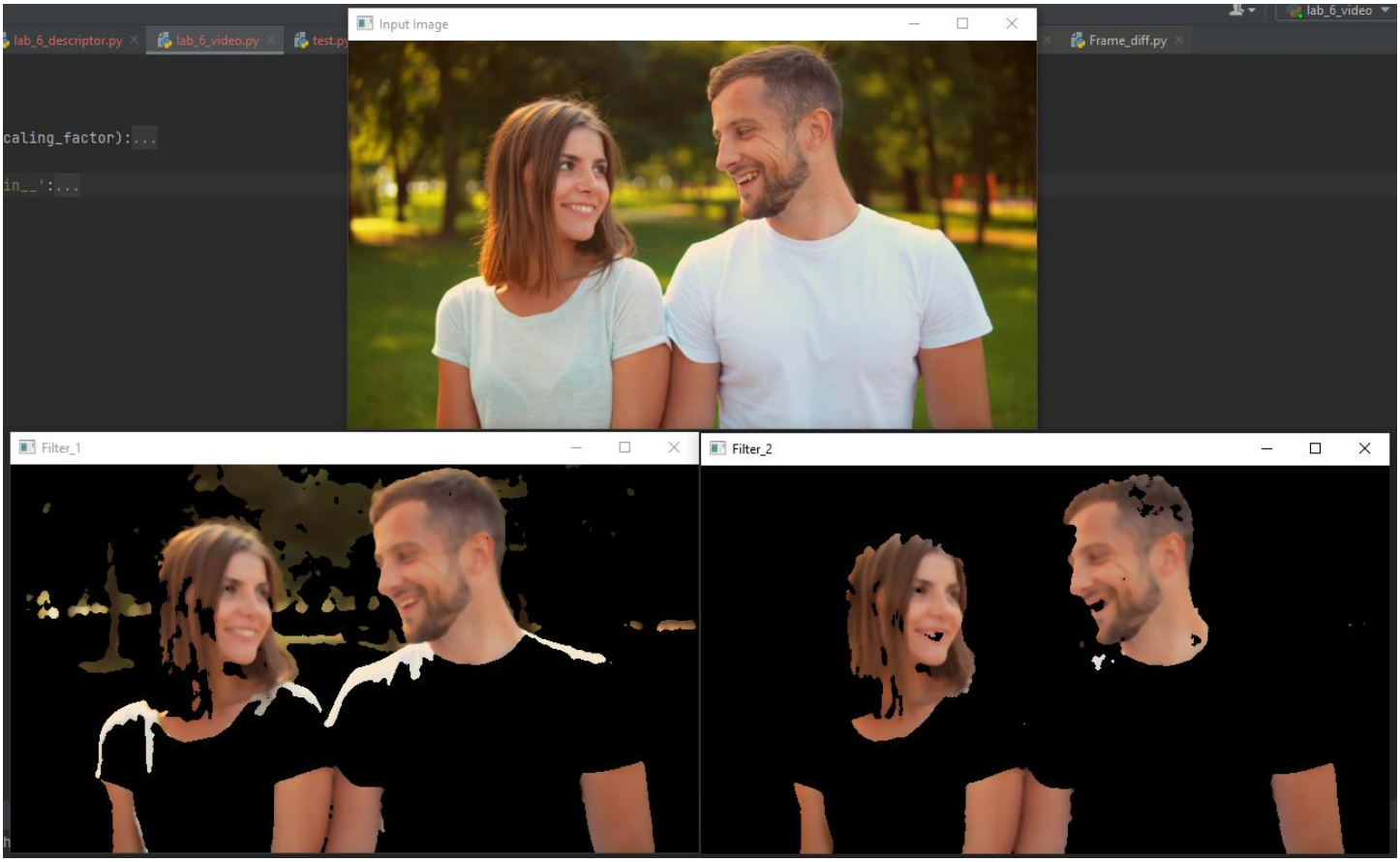


Рисунок 17 – Результат порівняння фільтрів на відео 1

Бачимо, що **для яскравих теплих відтінків шкіри краще підходить темний фільтр**. Тоді як світліший фільтр захоплює частини футболок та задній фон.

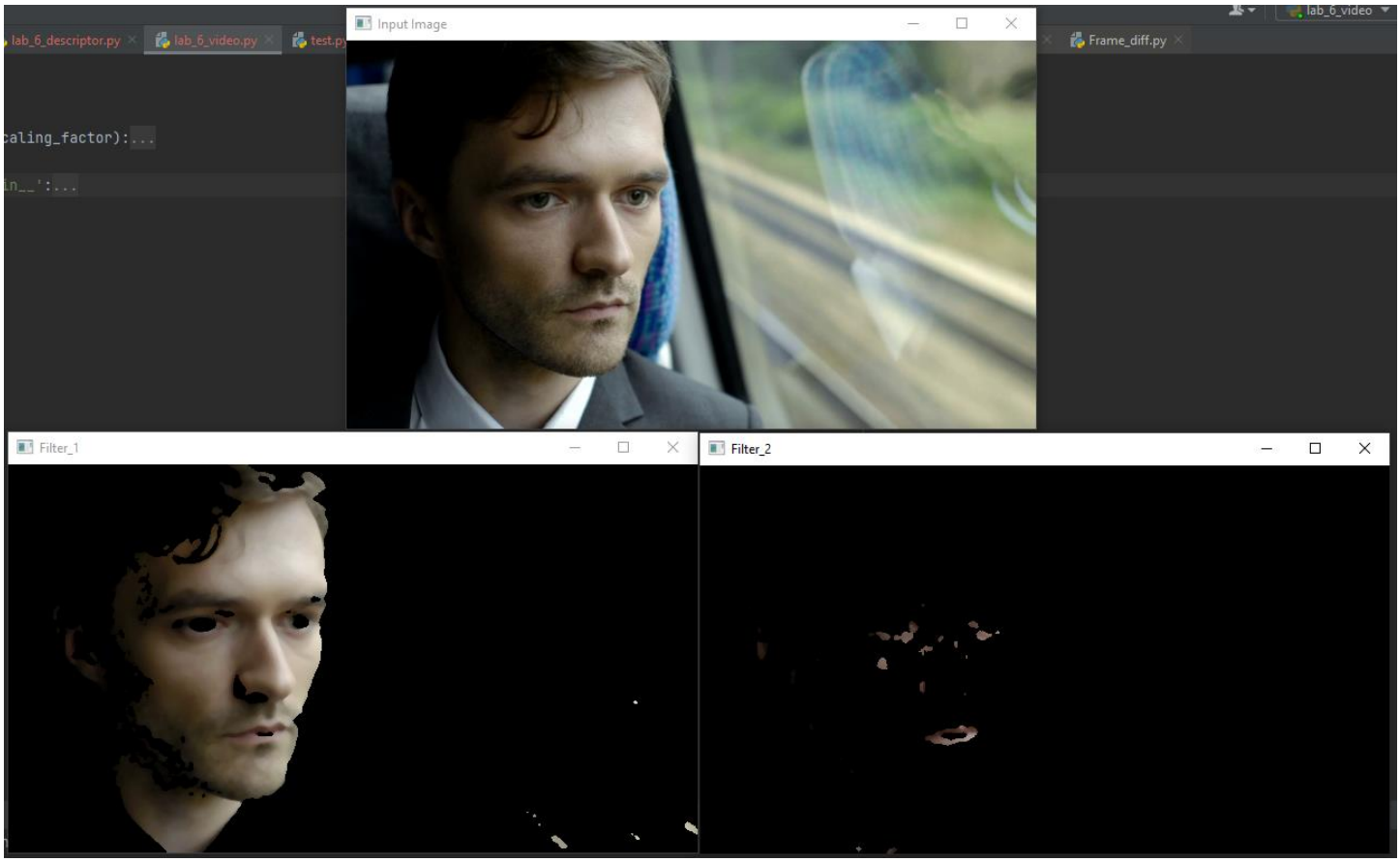


Рисунок 18 – Результат порівняння фільтрів на відео 2

Для **блідої шкіри при поганому освітленні підходить лише світлий фільтр**. Темний ідентифікує лише губи такої людини.

Спробуємо протестувати фільтри на відео, які не застосовувались для створення фільтрів.

Наступне відео містить людей різного типу шкіри, які міняються один за одним. На ньому буде добре провести оцінку фільтрів.

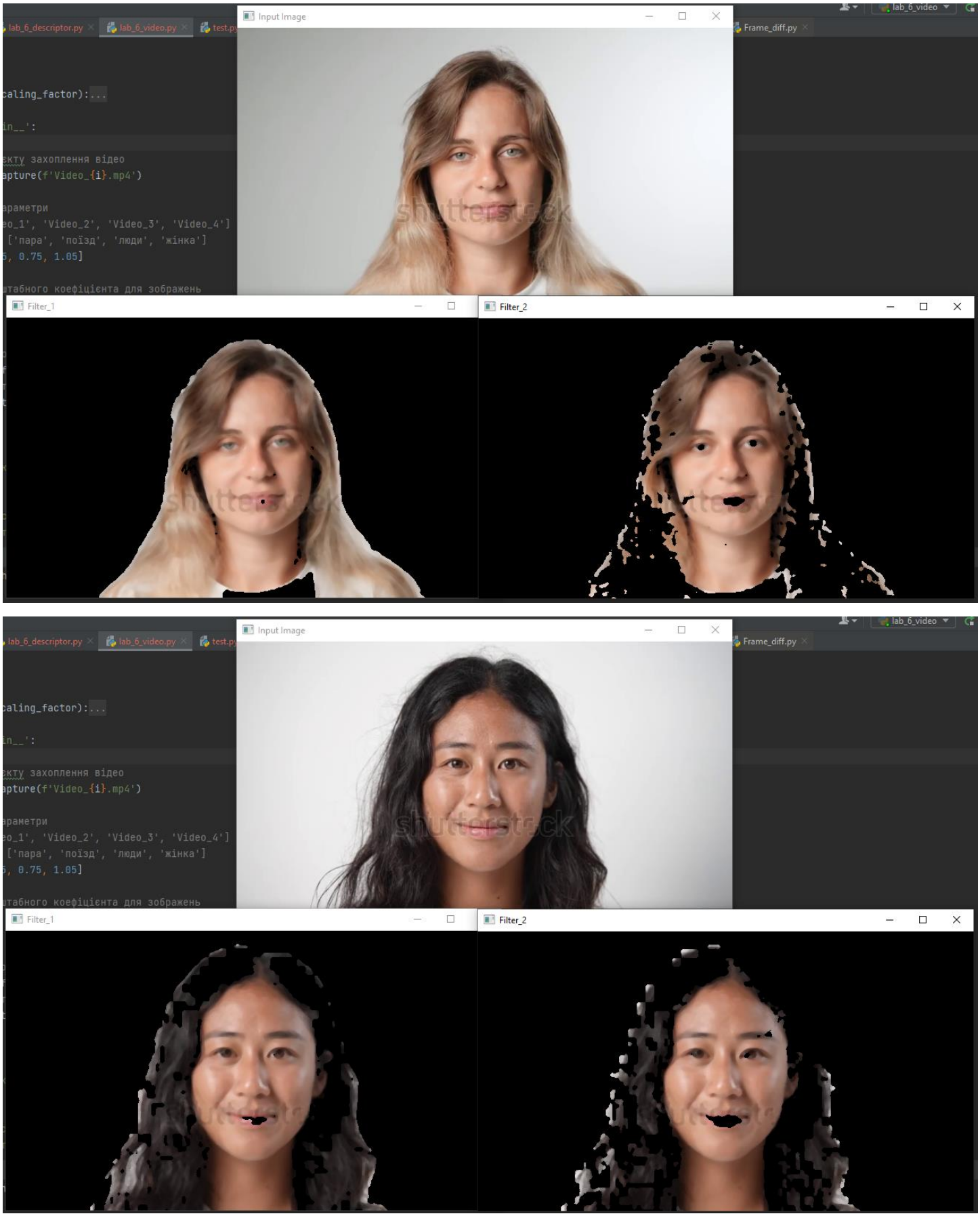


Рисунок 19 – Результат порівняння фільтрів на відео 3 (1)

Неочікувано, бачимо, що **фільтр для світлої шкіри показує себе краще в обох випадках** (бліда та смугла шкіра). Тоді як темніший фільтр відсікає забагато світлих зон обличчя та помилково додає білий колір з фону. Яскравіше це видно на наступному зображенні.

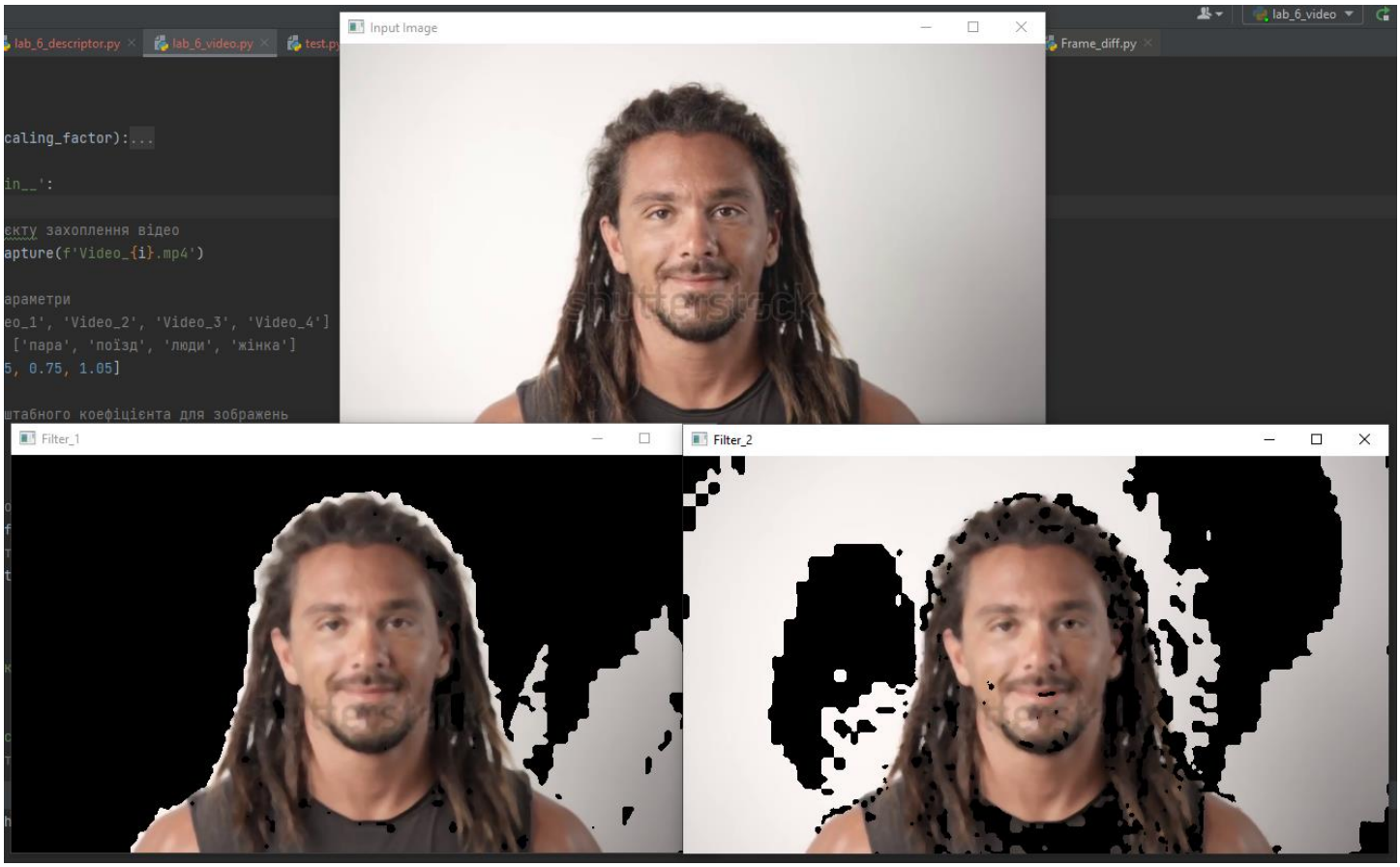


Рисунок 20 – Результат порівняння фільтрів на відео 3 (2)

Зрозуміло, що білий фон – ідеалізовані умови для відео зйомки, але очевидно, що темний фільтр його має відсікати. Тому **необхідно провести корекцію яскравості другого фільтру**.

Змінивши параметри фільтру на наступні, **отримаємо позитивні результати**.

```
# Фільтр 2
lower_2 = np.array([3, 30, 50])
upper_2 = np.array([11, 180, 255])
```

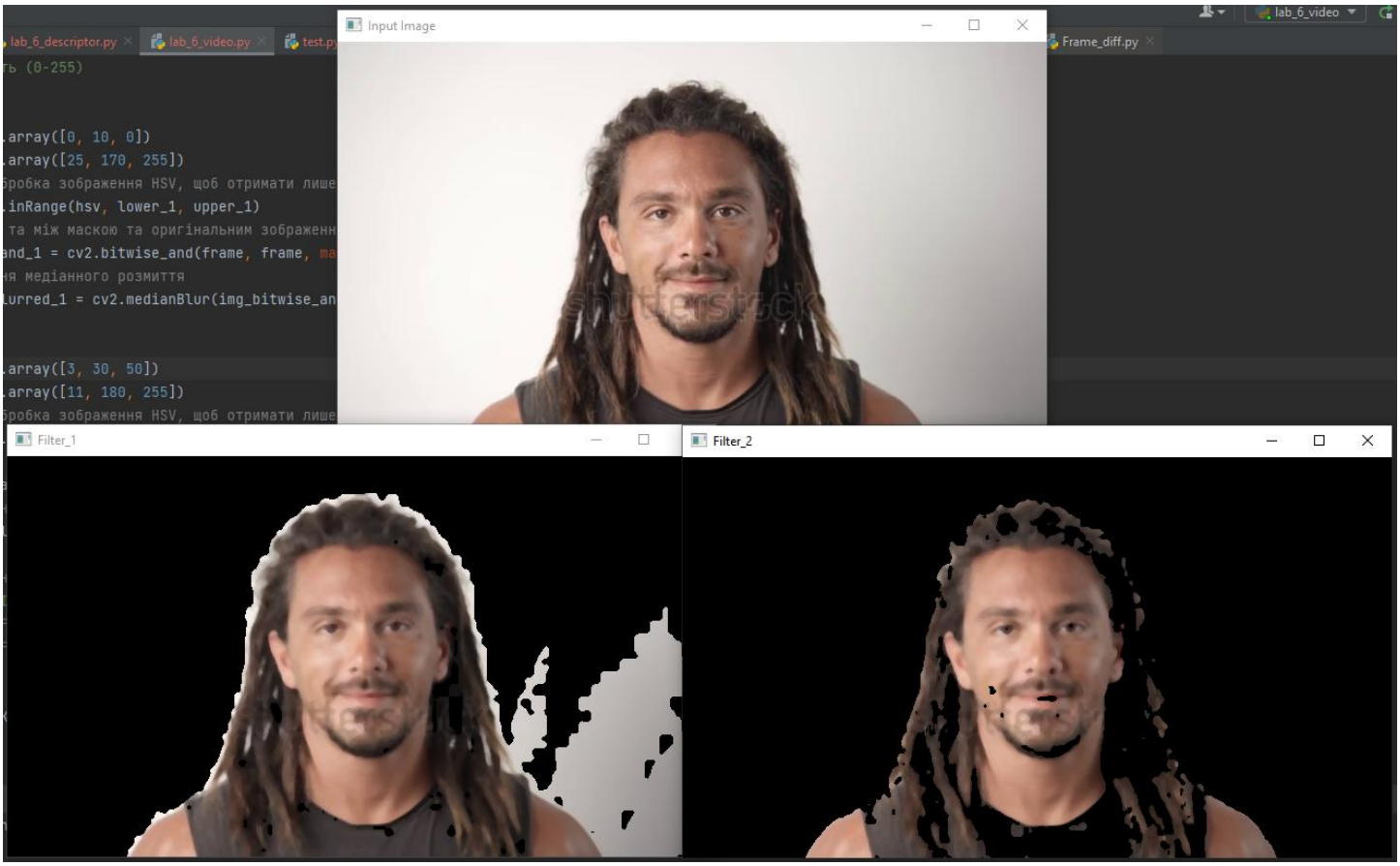


Рисунок 21 – Результат порівняння фільтрів на відео 3 (3)

У результаті бачимо, що **темний фільтр таки краще розпізнає людей з темною шкірою ніж світлий**.

Отже, задачу можна вважати успішно виконаною.

Лістинг коду:

```
import cv2
import numpy as np

# Захоплення кадра
def get_frame(cap, scaling_factor):
    # Прочитати поточний кадр з об'єкту захоплення відео
    _, frame = cap.read()
    # Змінити розмір зображення
    frame = cv2.resize(frame, None, fx=scaling_factor, fy=scaling_factor,
interpolation=cv2.INTER_AREA)

    return frame

if __name__ == '__main__':
    i = 3
    # Визначення об'єкту захоплення відео
    cap = cv2.VideoCapture(f'Video_{i}.mp4')
```

```
    # Опис відео і параметри
    # videos = ['Video_1', 'Video_2', 'Video_3', 'Video_4']
    # descriptions = ['пара', 'поїзд', 'люди', 'жінка']
    scale = [0.5, 0.5, 0.75, 1.05]

    # Визначення масштабного коефіцієнта для зображень
    scaling_factor = scale[i - 1]    # Постійне читання кадрів

    while True:
        # Отримати поточний кадр
        frame = get_frame(cap, scaling_factor)
        # Конвертувати зображення в простір кольорів HSV
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Визначення діапазону кольору шкіри в HSV
        '''
        HSV (відтінок, насиченість, яскравість)

            відтінок (0-360) - 0 червоний, 120 зелений, 240 синій
            насиченість (0-255)
            яскравість (0-255)
        '''
        # Фільтр 1
        lower_1 = np.array([0, 10, 0])
        upper_1 = np.array([25, 170, 255])
        # Порогова обробка зображення HSV, щоб отримати лише колір шкіри
        mask_1 = cv2.inRange(hsv, lower_1, upper_1)
        # Побітове І та між маскою та оригінальним зображенням
        img_bitwise_and_1 = cv2.bitwise_and(frame, frame, mask=mask_1)
        # Застосування медіанного розмиття
        img_median_blurred_1 = cv2.medianBlur(img_bitwise_and_1, 5)

        # Фільтр 2
        lower_2 = np.array([3, 30, 50])
        upper_2 = np.array([11, 180, 255])
        # Порогова обробка зображення HSV, щоб отримати лише колір шкіри
        mask_2 = cv2.inRange(hsv, lower_2, upper_2)
        # Побітове І та між маскою та оригінальним зображенням
        img_bitwise_and_2 = cv2.bitwise_and(frame, frame, mask=mask_2)
        # Застосування медіанного розмиття
        img_median_blurred_2 = cv2.medianBlur(img_bitwise_and_2, 5)

        # Показ вхідного зображення, зображення з контурами та обробленого зображення
        cv2.imshow('Input Image', frame)
        cv2.imshow('Filter_1', img_median_blurred_1)
        cv2.imshow('Filter_2', img_median_blurred_2)

        # Перевірка, чи натиснута клавіша 'Esc'
        c = cv2.waitKey(5)
        if c == 27:
            break

    cv2.destroyAllWindows()
```

2.3. Аналіз отриманих результатів

Виконано ПЕРШУ групу вимог:

- доповнено скрипт лабораторної роботи №5;
- реалізовано порівняння двох зображень за допомогою кутів Харріса і визначення особливих точок зображень за допомогою SIFT дескрипторів;
- додано оцінку ідентичності об’єктів на зображеннях;

– підтверджено рисунком.

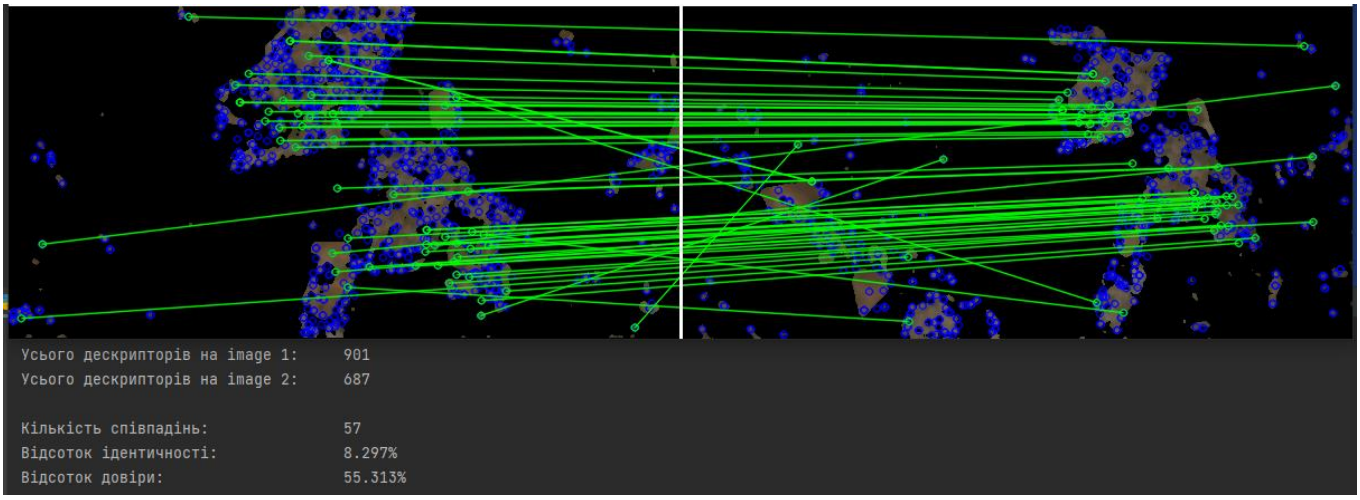


Рисунок 22 – Підтвердження виконання вимог 1

Виконано ТРЕТЮ групу вимог:

- реалізовано стеження за людиною за допомогою оцінки кольорових просторів гістограми яскравості;

– підтверджено рисунком.

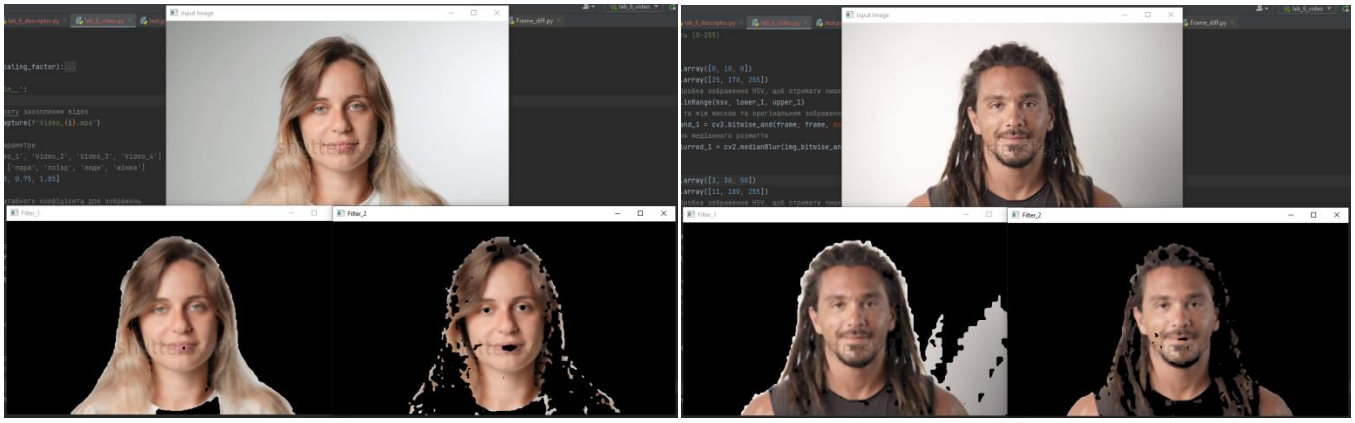


Рисунок 23 – Підтвердження виконання вимог 3

Висновок:

У результаті виконання лабораторної роботи отримано практичні та теоретичні навички порівняння цифрових зображень та стеження за об’єктами у відеопотоці.

Реалізовано програмний скрипт №1, який виконує оцінку ідентичності поселень зі знімків з супутника. Такий скрипт можна використовувати для того, щоб «зклеїти» декілька зображень з супутника у карту. Так, коли, ідентифікується поселення, яке вже було ідентифіковане йому може присвоюватися не новий ідентифікатор, а вже існуючий.

Реалізовано програмний скрипт №2, який відстежує людей у відеопотоці. Такий скрипт можна використовувати для відстеження рухів людей з невеликої відстані від камери у гейм індустрії, а також для накладання людського обличчя на аватар.