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

Кафедра інформаційних систем та технологій

Лабораторна робота №5

з дисципліни «Вступ до технології Data Science»

на тему

«РЕАЛІЗАЦІЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ (MACHINE LEARNING (ML))»

Виконала:
студентка групи ІС-12
Павлова Софія

Перевірив:
Баран Д. Р.

Київ – 2023

1. Постановка задачі

Мета роботи:

Виявити дослідити та узагальнити особливості аналізу даних з використанням методів та технологій машинного навчання (Machine Learning (ML)).

Завдання III рівня:

Реалізувати на вибір ТРИ з п’яти сформованих груп технічних вимог.

Група технічних вимог_3:

Підрахувати кількість об’єктів на обраному цифровому зображенні. Об’єкти, що підлягають обрахунку обрати самостійно. Зміст етапів попередньої обробки зображень (корекція кольору, фільтрація, векторизація, кластеризація) має буди результатом R&D процесів, що конкретизується обраним зображенням і об’єктами для підрахунку. Провести аналіз отриманих результатів, сформувати висновки.

Група технічних вимог_4:

Порівняти два зображення. Цифрові зображення та об’єкти для порівняння обрати самостійно. Зміст етапів попередньої обробки зображень (корекція кольору, фільтрація, векторизація, кластеризація (за необхідності)) має буди результатом R&D процесів, що конкретизується обраним зображенням і об’єктом для порівняння. Провести аналіз отриманих результатів, сформувати висновки.

Група технічних вимог_5:

Ідентифікувати об’єкти в обраному відеопотоці. Об’єкти, що підлягають ідентифікації обрати самостійно. Зміст етапів попередньої обробки відеопотоку (корекція кольору, фільтрація, векторизація, кластеризація (за необхідності)) має буди результатом R&D процесів, що конкретизується обраним відео і об’єктом ідентифікації. Провести аналіз отриманих результатів, сформувати висновки.

2. Виконання

2.1. Підрахунок кількості об’єктів на цифровому зображенні

Сформулюємо завдання з області **image recognition**.

Постановка задачі:

Наближається новий рік. Декоратори отримують багато замовлень на прикрашання новорічними прикрасами великих ялинок, висота яких більша 5м. Часто замовник просить прикрасити ялинку іграшками різної форми і вказує бажане співвідношення традиційних великих кульок на ялинці з прикрасами іншої форми.

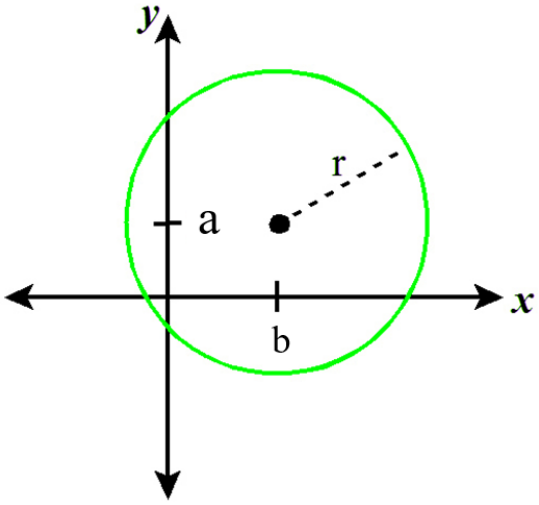
Декоратори намагалися рахувати кількість іграшок-кульок вручну, але в них це не дуже добре виходить, вони постійно збиваються і починають спочатку.

Для полегшення праці декораторам необхідно програмне забезпечення, що за допомогою камери розпізнає серед інших іграшок «кульки» і підраховує їх кількість.

Математична модель:

Отже наша задача зводиться до виявлення кіл або круглих об'єктів на зображенні, їх візуалізація та підрахунок кількості розпізнаних об’єктів.

Коло можна описати наступним рівнянням:



$$(x-a)^2+(y-b)^2=r^2$$

Рисунок 1 – Рівняння кола

Для виявлення кіл можна зафіксувати точку (x, y). Тепер нам потрібно знайти 3 параметри: a, b та r. Щоб знайти можливі кола, алгоритм використовує 3-D матрицю, яка називається «Матриця акумулятора», для зберігання потенційних значень a, b та r. Значення a (x-координата центру) може коливатися від 3 до рядків, b (y-координата центру) може коливатися від 1 до $cols$, а r може коливатися від 1 до $maxRadius = \sqrt{rows^2 + cols^2}$.

Для цього доцільно використати бібліотеку **OpenCV** і наступний **алгоритм виявлення кіл**:

1. **Ініціалізація «матриці акумулятора»**

Ініціалізуємо матрицю розмірності: $rows * cols * maxRadius$ нулями.

2. **Попередня обробка зображення**

Застосуємо до зображення розмиття, відтінки сірого та детектор країв. Це робиться для того, щоб кола відображалися як затемнені краї зображення.

3. **Прохід по точках**

Проходячи по точках виберемо точку x_i на зображенні.

4. **Фіксація r і зациклення через a і b**

Використаємо подвійний вкладений цикл, щоб знайти значення r, змінюючи a та b в заданих діапазонах.

5. **Голосування**

Виберемо точки «матриці акумулятора» з максимальним значенням. Це сильні точки, які вказують на існування кола з параметрами a, b та r. Це дає нам простір кіл Хафа (Hough).

6. **Знаходження кіл**

Нарешті, використовуючи вищевказані кола як кола кандидатів, проголосуйте відповідно до зображення. Коло з максимальною кількістю голосів у матриці акумулятора дає нам коло.

Функція ***HoughCircles()*** в **OpenCV** має наступні параметри, які можуть бути змінені відповідно до зображення.

OpenCV має розширену реалізацію *HOUGH_GRADIENT*, яка використовує градієнт країв замість заповнення всієї матриці 3D-акумулятора, тим самим прискорюючи процес.

dp: це відношення роздільної здатності вихідного зображення до матриці акумулятора.

minDist: цей параметр контролює мінімальну відстань між виявленими колами.

Param1: для виявлення країв потрібні два параметри — $minVal$ і $maxVal$. $Param1$ є вищим порогом з двох. Другий встановлюється як $Param1/2$.

Param2: це поріг акумулятора для виявлених кіл кандидата. Збільшивши це порогове значення, ми можемо гарантувати, що будуть повернуті лише найкращі кола, що відповідають більшим значенням акумулятора.

minRadius: мінімальний радіус кола.

maxRadius: максимальний радіус кола.

На основі цього поетапно напишемо програму.

Навчимо спочатку програму розпізнавати круглі об’єкти, а потім протестуємо її на двох картинках ялинкових прикрас.

Для зручності користування програмою передбачимо можливість вибору користувачем зображення для розпізнавання зі списку.

Зчитування зображення

Зчитаємо та виведемо зображення для візуалізації відправної точки.

Лістинг коду:

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

# Зчитування зображення
def image_read (file_name):
    image = cv2.imread(file_name, cv2.IMREAD_COLOR)
```

```
    plt.imshow(image)
    plt.show()

    return image

if __name__ == '__main__':

    images = ['circles.jpg', 'jalynka.jpg', 'jalynka_2.jpg']
    descriptions = ['круги', 'ялинкові прикраси', 'ялинкові прикраси 2']
    processed_images = ['circles_filtred.jpg', 'jalynka_filtred.jpg',
'jalynka_2_filtred.jpg']
    results = ['circles_reconition.jpg', 'jalynka_reconition.jpg',
'jalynka_2_reconition.jpg']

    # Вибір зображення
    print('Оберіть зображення для image recognition:')
    for i in range(len(descriptions)):
        print(i + 1, '-', descriptions[i])
    data_mode = int(input('mode:'))
    # Якщо джерело даних існує
    if data_mode in range(1, len(images) + 1):

        image = image_read(images[data_mode - 1])
```

Результат:

Оберемо спочатку зображення кругів для тестування роботи скрипту.

```
Оберіть зображення для image recognition:
1 - круги
2 - ялинкові прикраси
3 - ялинкові прикраси 2
mode:1
```

Рисунок 2 – Вибір зображення зі списку

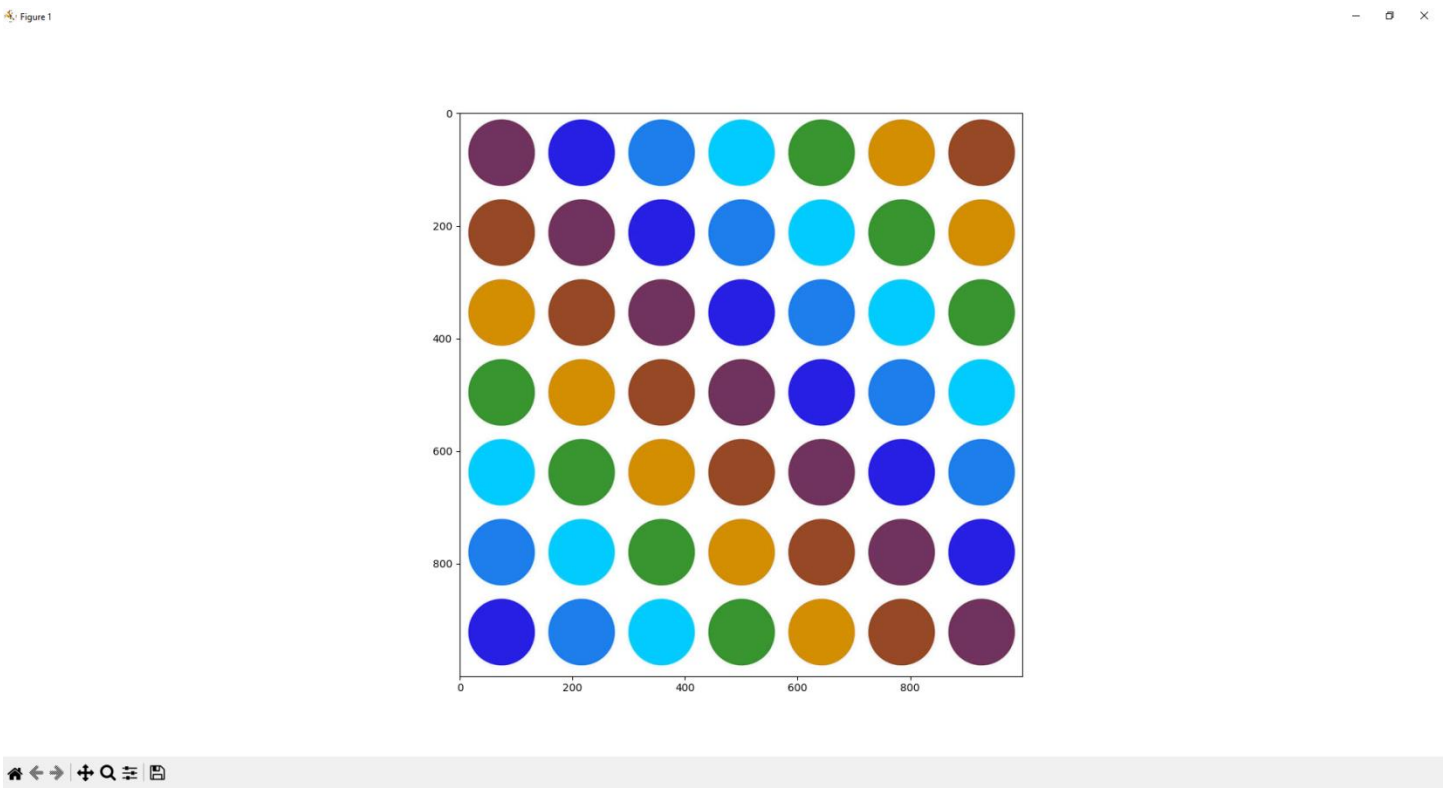


Рисунок 3 – Зчитане зображення **circles.jpg**

Обробка зображення

Перетворимо зображення в градації сірого та розмиємо ядром 3*3 з метою виділення контурів.

Лістинг коду:

```
[...]

# Обробка зображення
def image_processing (file_name, image):
    # Перетворити в градації сірого
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Розмиття за допомогою ядра 3 * 3
    gray_blurred = cv2.blur(gray, (3, 3))
    cv2.imwrite(file_name, gray_blurred)
    plt.imshow(gray_blurred)
    plt.show()

    return gray_blurred


if __name__ == '__main__':

    [...]
```

```
    # Вибір зображення
    [...]
    # Якщо джерело даних існує
    if data_mode in range(1, len(images) + 1):

        [...]
        processed_image = image_processing(processed_images[data_mode - 1], image)
```

Результат:

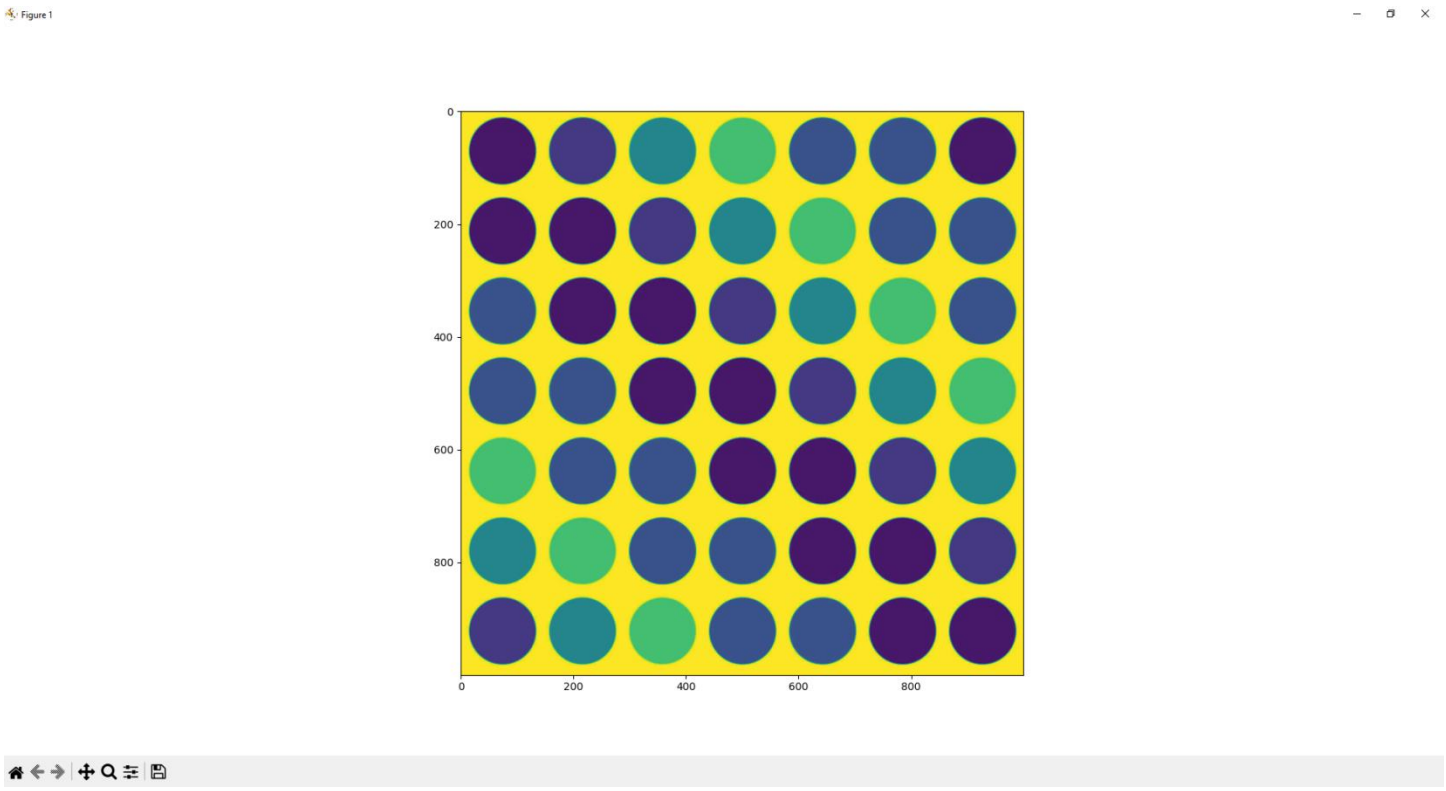


Рисунок 4 – Результат обробки зображення **circles.jpg**

Розпізнавання кіл

Напишемо функцію, яка знаходитиме та рахуватиме кількість розпізнаних кіл. Важливо зазначити, що для кожного зображення коефіцієнти функції *HoughCircles()*, про які йшлося раніше відрізнятимуться.

Для зображення **circles.jpg** використаємо **param1=10, param2=100, minRadius=1, maxRadius=400**.

Лістинг коду:

```
[...]

# Розпізнавання кругів на зображеннях
def circle_recognition(image, processed_image, file_name, var):
    # Застосування трансформації Хафа до розмитого зображення

    # circles
    if (var == 1):
        detected_circles = cv2.HoughCircles(processed_image,
                                            cv2.HOUGH_GRADIENT, 1, 20, param1=10,
                                            param2=100, minRadius=1, maxRadius=400)
    # jalynka
    elif (var == 2):
        detected_circles = cv2.HoughCircles(processed_image,
                                            cv2.HOUGH_GRADIENT, 1, 20, param1=250,
                                            param2=55, minRadius=1, maxRadius=400)
    # jalynka_2
    else:
        detected_circles = cv2.HoughCircles(processed_image,
                                            cv2.HOUGH_GRADIENT, 1, 20, param1=250,
                                            param2=100, minRadius=1, maxRadius=400)

    # Малювання виявлених кіл
    if detected_circles is not None:
        # Перетворення параметрів кола a, b і r на цілі числа
        detected_circles = np.uint16(np.around(detected_circles))
        for pt in detected_circles[0, :]:
            a, b, r = pt[0], pt[1], pt[2]
            # Малювання окружності кола
            cv2.circle(image, (a, b), r, (0, 255, 0), 7)
            # Малювання маленького кола (радіусом 1), щоб показати центр
            cv2.circle(image, (a, b), 1, (0, 0, 255), 10)
        print("Знайдено {0} круглих об'єктів".format(len(detected_circles[0, :])))
        cv2.imwrite(file_name, image)
        plt.imshow(image)
        plt.show()

    return

if __name__ == '__main__':

    [...]

    # Вибір зображення
    [...]
    # Якщо джерело даних існує
    if data_mode in range(1, len(images) + 1):

        [...]
        circle_recognition(image, processed_image, results[data_mode - 1], data_mode)
```

Результат:

```
Знайдено 49 круглих об'єктів
```

Рисунок 5 – Кількість розпізнаних кіл на зображенні **circles.jpg**

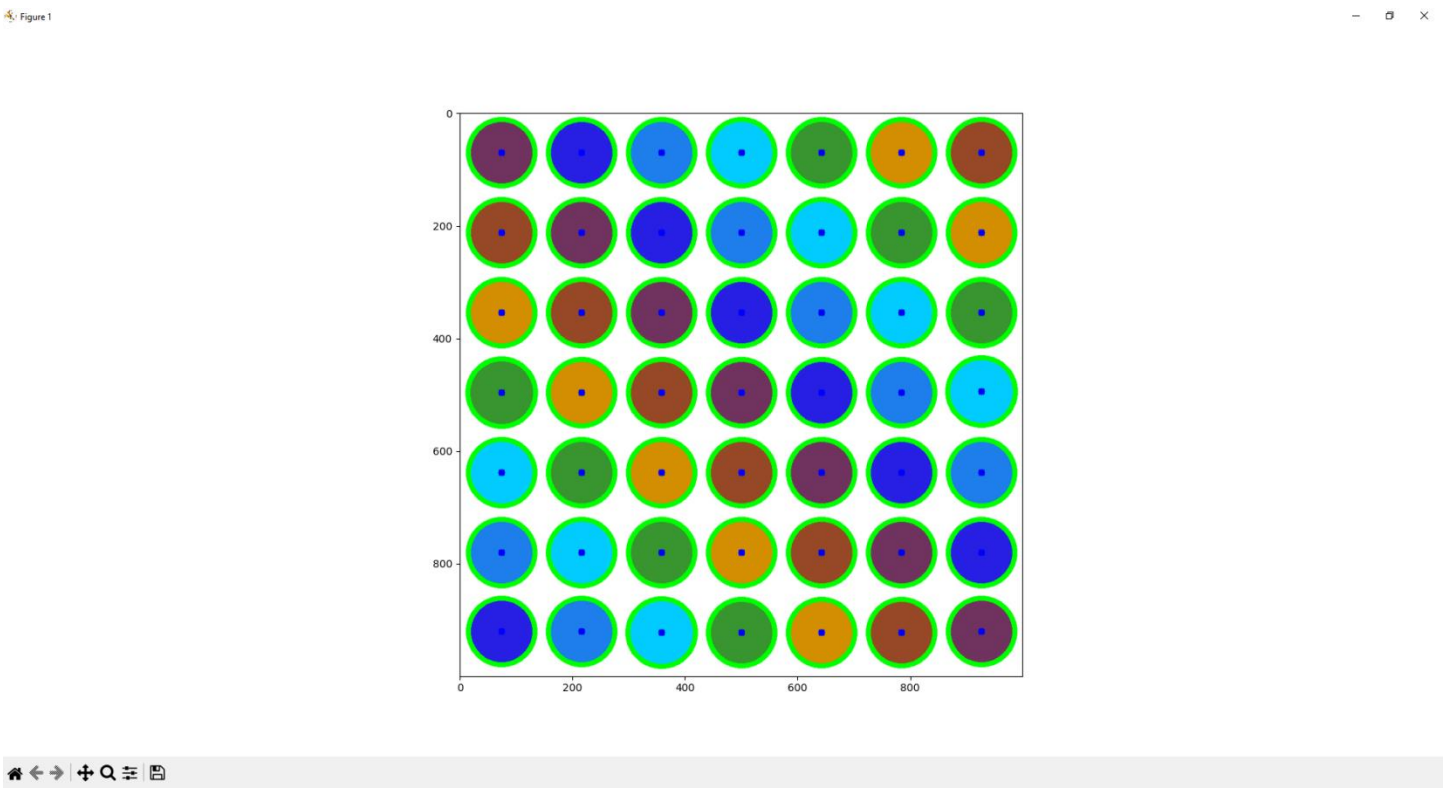


Рисунок 6 – Результат розпізнавання кіл на зображенні **circles.jpg**

Бачимо, що програма чудово справилась з поставленою задачею на тестовому зображенні. Перевіримо її працездатність на зображеннях **jalynka.jpg** та **jalynka_2.jpg**.

Тестування

Для зображення **jalynka.jpg** використаємо **param1=250, param2=55, minRadius=1, maxRadius=400**.

Результат:

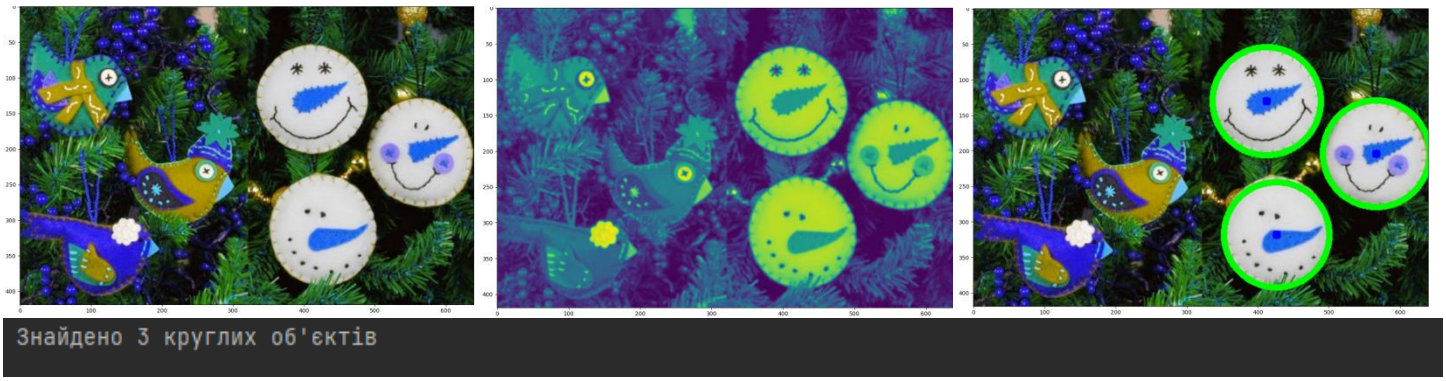


Рисунок 7 – Результат розпізнавання кіл на зображенні **jalynka.jpg**

Для зображення **jalynka_2.jpg** використаємо **param1=250, param2=100, minRadius=1, maxRadius=400**.

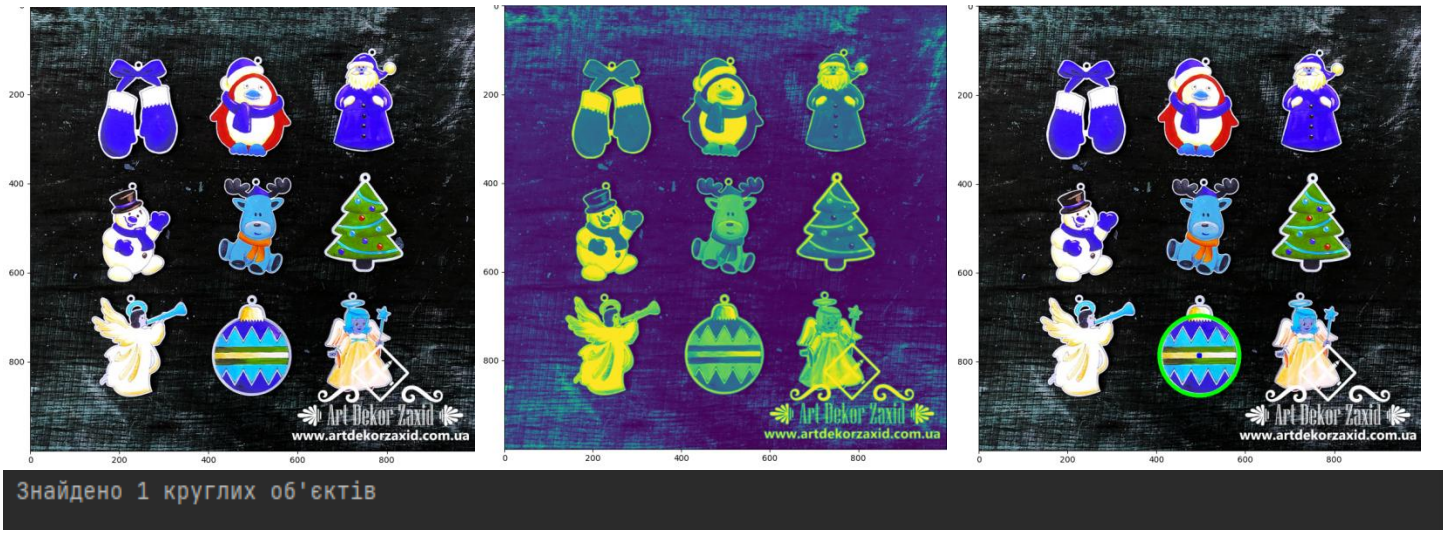


Рисунок 8 – Результат розпізнавання кіл на зображенні **jalynka_2.jpg**

Бачимо, що програма чудово розпізнає новорічні прикраси у вигляді куль. Утім кожне зображення потребує окремого налаштування параметрів функції Хаффа.

2.2. Порівняння двох зображень

Сформулюємо завдання з області **image description**.

Постановка задачі:

Українські військові відвоювали Маріуполь, але він весь лежить в руїнах. Утім ще залишились будівлі, обриси яких можна впізнати.

Незважаючи на те, що місто відвойовано, пересуватися по ньому ще дуже небезпечно. Тому єдина можливість оглянути ситуацію в місті – зробити обліт міста дроном.

Військові запросили програмне забезпечення, за допомогою якого дрон зможе порівняти довоєнні фотографії історичних пам’яток міста з їх теперішнім станом.

Математична модель:

Фактично задача зводиться до виявлення у двох зображень ознак та їх порівняння. Так як будівлі зазвичай мають прямокутні форми, серед ознак нас цікавлять – кути. Для цього доцільно використати **метод виявлення кутів Харріса** (Harris) та **дескриптор масштабно-інваріантного перетворення ознак (SIFT)** з бібліотекою **OpenCV**.

Але перед цим варто розібратися з термінологією.

Алгоритми функцій поділяються на дві категорії: детектори ознак і дескриптори ознак.

Детектор ознак знаходить цікаві області на зображенні. Вхідними даними в детектор ознак є зображення, а вихідними — піксельні координати значних областей зображення.

Дескриптор функції кодує цю функцію в числовий «відбиток». Опис об'єкта робить об'єкт унікальним для ідентифікації серед інших об'єктів на зображенні.

Розглянемо алгоритм Харріса.

Алгоритм виявлення кутів Харріса має наступні кроки:

1. Визначення, яка частина зображення має велику різницю інтенсивності, оскільки кути мають великі відмінності в інтенсивності. Це досягається шляхом переміщення ковзного вікна по всьому зображенню.
2. Для кожного визначеного вікна обчислення значення показника R.
3. Застосування $threshold$ до $score$ та позначення кутів.

Недоліком цього алгоритму є те, що він не підходить для виділення спільних ознак у предметів, які крутять. Тобто цей алгоритм не знайде спільні ознаки у будівлі знятої з різних ракурсів. Тому для наших цілей можна використати **дескриптор SIFT для заданих функцій на основі виявлення кутів Харріса**.

На основі цього поетапно напишемо програму.

Реалізуємо спочатку детектор кутів і протестуємо його на зображеннях Драм театру в Маруполі, взятих з *Google Maps 3D View*.

Для зручності користування програмою передбачимо декілька режимів роботи програми і дамо користувачу можливість вибору, з якого режиму запускатись.

Детектор кутів Харріса

У якості зображень для зіставлення було обрано 2 зображення Драм театру в Маріуполі з різних ракурсів, менший фрагмент театру – арку над головним входом та проекцію зображення драмтеатру на кружку.

Лістинг коду:

```
import numpy as np
import cv2 as cv

# Детектор кутів Харріса
def harris_corner_detector (filename, result_file):
    img = cv.imread(filename)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv.cornerHarris(gray, 2, 3, 0.04)

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

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

    cv.imwrite(result_file, img)
    cv.imshow('Harris_Corner_Detector', img)
    if cv.waitKey(0) & 0xff == 27:
        cv.destroyAllWindows()

    return

if __name__ == '__main__':

    descriptions = ['порівняння театра з зображенням на чашці', 'порівняння ракурса 1
театра зі шматком', 'порівняння ракурса 2 театра зі шматком']

    # Вибір режиму роботи програми
    print('Оберіть задачу для image descriptor:')
    for i in range(len(descriptions)):
        print(i + 1, '-', descriptions[i])
    data_mode = int(input('mode:'))
    # Якщо джерело даних існує
```

```
    if data_mode in range(1, len(descriptions) + 1):
        # Театр & чашка
        if (data_mode == 1):
            # Детектор кутів Харріса
            harris_corner_detector('theater_1.png', 'theater_1_harris.png')
            harris_corner_detector('cup.png', 'cup_harris.png')
        # Театр & шматок
        elif (data_mode == 2):
            # Детектор кутів Харріса
            harris_corner_detector('theater_1.png', 'theater_1_harris.png')
            harris_corner_detector('theater_1_cut.png', 'theater_1_cut_harris.png')
        # Театр 2 & шматок
        else:
            # Детектор кутів Харріса
            harris_corner_detector('theater_2.png', 'theater_2_harris.png')
            harris_corner_detector('theater_1_cut.png', 'theater_1_cut_harris.png')
```

Результат:

У результаті отримуємо наступні розпізнані кути на зображеннях.

Рисунок 9 – Результат детектора кутів Харріса на зображенні **theater_1.png**

Рисунок 10 – Результат детектора кутів Харріса на зображенні **theater_2.png**

Рисунок 11 – Результат детектора кутів Харріса на зображенні **theater_1_cut.png**

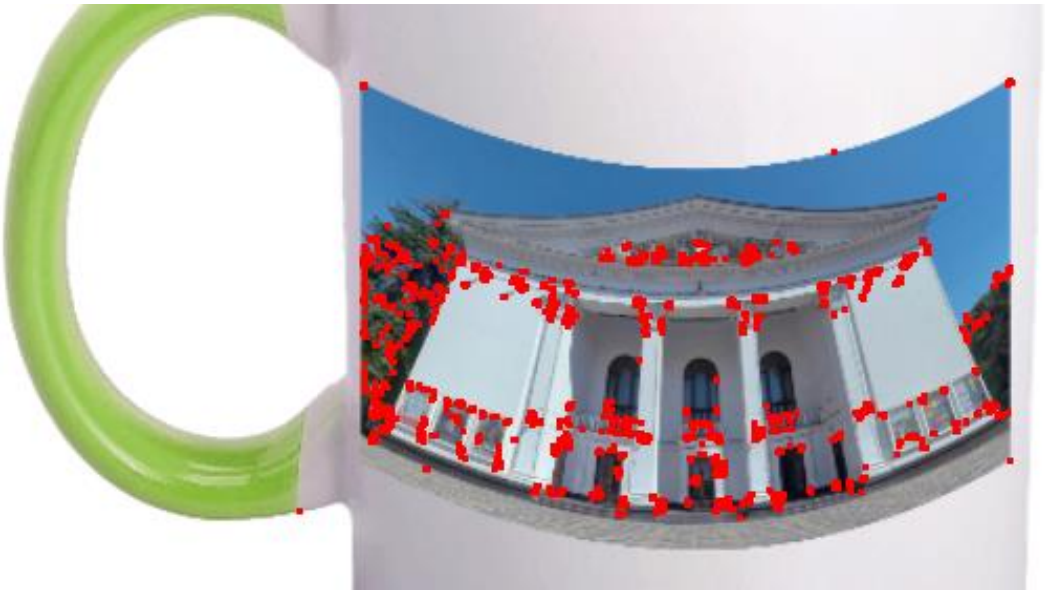


Рисунок 12 – Результат детектора кутів Харріса на зображенні **cup.png**

Бачимо, що алгоритм чудово визначає кути, але йому важко справлятись на неконтрастних, затемнених зображеннях.

Перейдемо до реалізації дескриптора SIFT.

Дескриптор SIFT на основі алгоритму виявлення кутів Харріса

Напишемо такий дескриптор, у основі якого лежить розрахунок дескриптора SIFT для ключових точок (кутів), виявлених алгоритмом Харріса.

Лістинг коду:

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

    def harris (img):
        gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        gray_img = np.float32(gray_img)
        dst = cv.cornerHarris(gray_img, 2, 3, 0.04)
        result_img = img.copy()  # Бекапна копія зображення

        # 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 = [cv.KeyPoint(float(x[1]), float(x[0]), 13) for x in keypoints]

        return (keypoints, result_img)

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

    cv.imwrite(result_file, img)
    cv.imshow('SIFT', img)
    if cv.waitKey(0) & 0xff == 27:
        cv.destroyAllWindows()

    return

if __name__ == '__main__':

    [...]

    # Вибір режиму роботи програми
    [...]
    # Якщо джерело даних існує
    if data_mode in range(1, len(descriptions) + 1):
```

```
        # Театр & чашка
        if (data_mode == 1):
            [...]
            # Дескриптори SIFT для кутів Харріса
            sift_descriptors_on_harris('theater_1.png', 'theater_1_sift.png')
            sift_descriptors_on_harris('cup.png', 'cup_sift.png')
        # Театр & шматок
        elif (data_mode == 2):
            [...]
            # Дескриптори SIFT для кутів Харріса
            sift_descriptors_on_harris('theater_1.png', 'theater_1_sift.png')
            sift_descriptors_on_harris('theater_1_cut.png', 'theater_1_cut_sift.png')
        # Театр 2 & шматок
        else:
            [...]
            # Дескриптори SIFT для кутів Харріса
            sift_descriptors_on_harris('theater_2.png', 'theater_2_sift.png')
            sift_descriptors_on_harris('theater_1_cut.png', 'theater_1_cut_sift.png')
```

Результат:

У результаті отримуємо наступні значення ознак SIFT на розпізнаних кутах на зображеннях.

Рисунок 13 – Результат дескриптора SIFT на зображенні **theater_1.png**

Рисунок 14 – Результат дескриптора SIFT на зображенні **theater_2.png**

Рисунок 15 – Результат дескриптора SIFT на зображенні **theater_1_cut.png**

Рисунок 16 – Результат дескриптора SIFT на зображенні **cup.png**

Кожне з цих кіл вказує на розмір цього об'єкта. Лінія всередині кола вказує на орієнтацію об'єкта. Так як наші ознаки – кути, бачимо, що їх розмір невеликий. Отже алгоритм чудово справляється зі своєю задачею.

Перейдемо до реалізації зіставлення ознак на основі дескриптора SIFT.

Порівняння двох зображень

Лістинг коду:

```
# Порівняння двох зображень
def sift_feature_matching (filename_1, filename_2, result_file):

    img1 = cv.imread(filename_1, cv.IMREAD_GRAYSCALE)
    img2 = cv.imread(filename_2, cv.IMREAD_GRAYSCALE)

    # Запуск детектора SIFT
    sift = cv.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 = cv.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des1, des2, k=2)

    # Створення маски
    matchesMask = [[0, 0] for i in range(len(matches))]

    # Тест співвідношення
    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=cv.DrawMatchesFlags_DEFAULT)

    img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches, None, **draw_params)

    cv.imwrite(result_file, img3)
    cv.imshow('sift_feature_matching', img3)
    if cv.waitKey(0) & 0xff == 27:
        cv.destroyAllWindows()

    return

if __name__ == '__main__':

    [...]
```

```
    # Вибір режиму роботи програми
    [...]
    # Якщо джерело даних існує
    if data_mode in range(1, len(descriptions) + 1):
        # Театр & чашка
        if (data_mode == 1):
            [...]
            # Співставлення ознак
            sift_feature_matching('theater_1.png', 'cup.png', 'theater_1_cup.png')
        # Театр & шматок
        elif (data_mode == 2):
            [...]
            # Співставлення ознак
            sift_feature_matching('theater_1.png', 'theater_1_cut.png',
'theater_1_theater_1_cut.png')
        # Театр 2 & шматок
        else:
            [...]
            # Співставлення ознак
            sift_feature_matching('theater_2.png', 'theater_1_cut.png',
'theater_2_theater_1_cut.png')
```

Результат:

У результаті отримуємо наступні результати порівняння зображень. Сині точки – розпізнані кути, а зелені лінії – розпізнані спільні ознаки.

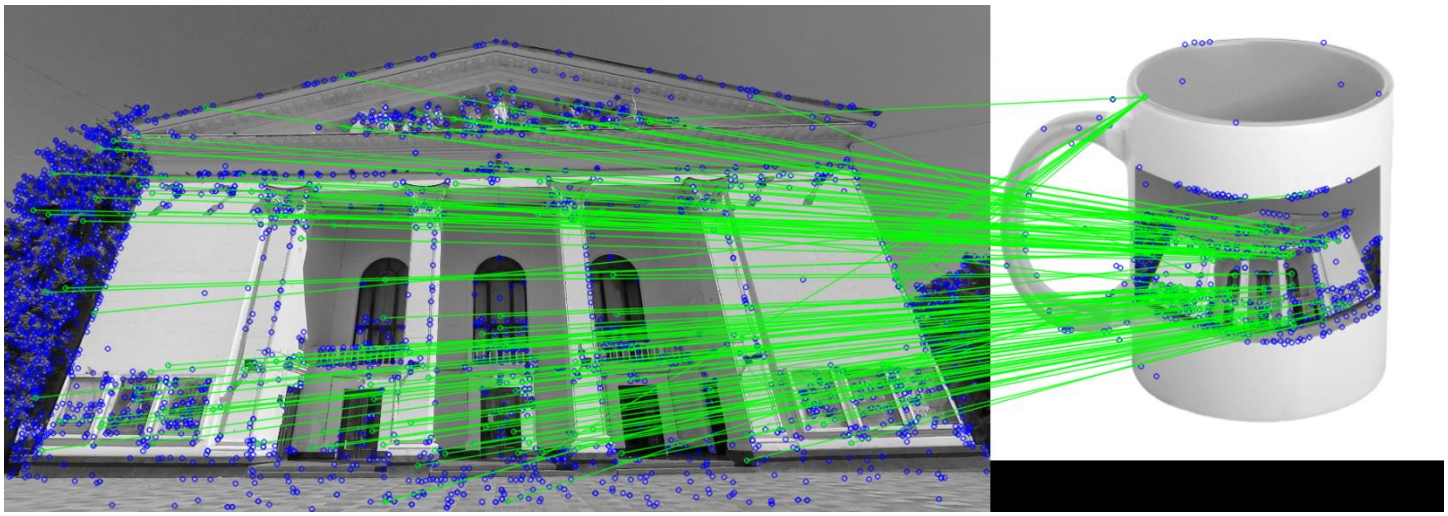


Рисунок 17 – Результат порівняння зображень **theater_1.png** та **cup.png**

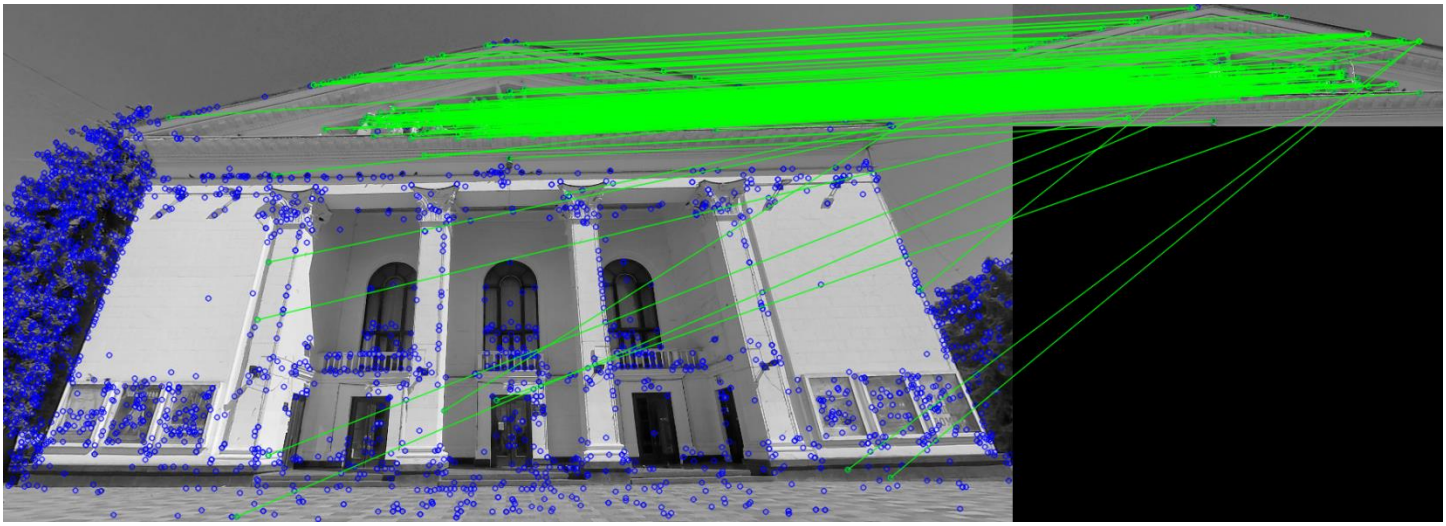


Рисунок 18 – Результат порівняння зображень **theater_1.png** та **theater_1_cut.png**

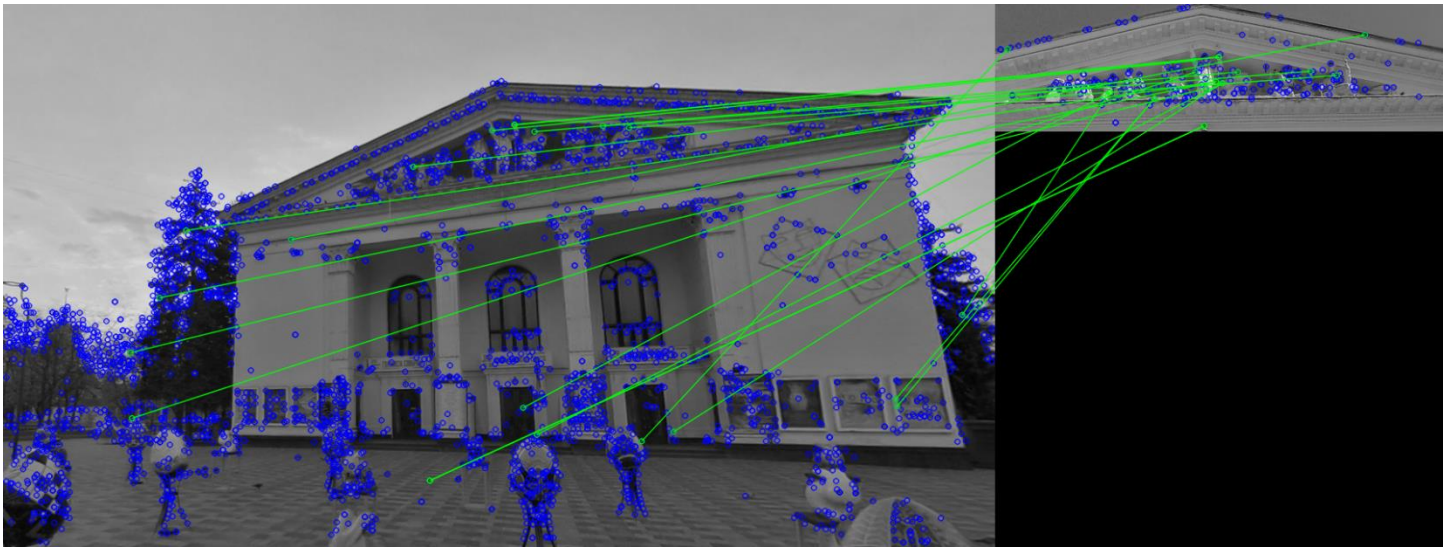


Рисунок 19 – Результат порівняння зображень **theater_2.png** та **theater_1_cut.png**

Бачимо, що програма чудово розпізнала проекцію зображення на чашку, а отже й впорається з викривленням зображення через камеру дрону. Також програма розпізнала шматок будівлі для другого зображення, утім має деякі неточності. І врешті решт програма також розпізнала схожість шматка будівлі з іншим ракурсом на ту ж саму будівлю. Можемо помітити, що програма знаходить й хибні спів падіння, але це можна списати на якість зображення.

Отже з поставленою задачею ми впорались, але ще є куди модифікувати код.

2.3. Ідентифікація об’єктів у відеопотоці

Сформулюємо завдання з області **object tracking**.

Постановка задачі:

На вулицях Нью-Йорка почастішали крадіжки. Поліція не встигає розгрібати заяви від обурених громадян. Їм потрібна допомога.

Поліцейські просять розробити програму, яка зможе відстежувати крадія по камерах відеоспостереження, встановлених на вулицях. Це в рази пришвидшило б їх роботу.

Математична модель:

Вважатимемо, що поліцейські вже мають заздалегідь навчений класифікатор, що дозволяє точно визначити крадія серед натовпу людей на вулиці. Наша задача фактично зводиться до відстеження цього об’єкту. У нашій моделі будемо використовувати рамку, намальовану від руки замість цього класифікатора.

Для вирішення цього завдання доцільно використати бібліотеку **OpenCV**, яка має багато алгоритмів для object tracking.

Наприклад:

1. ***MeanShift***: непараметричний алгоритм, який використовує гістограму кольорів для відстеження об'єктів.
2. ***CamShift***: розширення *MeanShift*, яке адаптується до змін розміру та орієнтації об'єкта.
3. ***KCF*** (Kernelized Correlation Filter): алгоритм на основі кореляції, який використовує функцію ядра для відображення об'єкта та навколишньої його області.
4. ***MOSSE*** (Minimum Output Sum of Squared Error): алгоритм відстеження об'єктів у реальному часі, який використовує циркулярну структуру для зниження обчислювальних витрат.

5. ***CSRT*** (Channel and Spatial Reliability Tracking): розширення алгоритму *KCF*, який використовує просторову та кольорову інформацію для покращення продуктивності відстеження.

Загальний **принцип роботи** кожного з цих алгоритмів наступний:

1. Зчитування відео.
2. Зчитування першого кадру.
3. Встановлення регіону інтересу (рамки).
4. Оновлення трекера кадр за кадром.
5. Малювання рамки відстежуваного об’єкту на кожному кадрі.
6. Запис кожного кадру у результуюче відео.

Для нашої задачі реалізуємо останні 3 алгоритми з переліку вище і порівняємо їх ефективність.

Напишемо програму таким чином, щоб користувач міг обирати джерело даних (у нашій задачі переключатися між камерами) та алгоритм відстеження. Зробимо так, щоб програма зберігала вихідний відео файл.

Алгоритм MOSSE для відстеження об'єктів

MOSSE є менш витратним з точки зору обчислень, ніж інші алгоритми, такі як *MeanShift* і *CamShift*. Тому реалізуємо його для нашої задачі.

<u>Лістинг коду:</u>

```
import cv2

videos = ['V_1', 'V_2', 'V_3', 'V_4', 'V_5', 'V_6', 'V_7']
descriptions = ['чоловік', 'поїздка в транспорті', 'віддалення', 'команда',
'автомагістраль', 'пішохідний перехід', 'собака']
algs = ['MOSSE', 'KCF', 'CSRT']

# Вибір відео
print('Оберіть відео для object tracking:')
for i in range(len(descriptions)):
    print(i + 1, '-', descriptions[i])
data_mode = int(input('mode:'))
```

```
# Якщо джерело даних існує
if data_mode in range(1, len(videos) + 1):

    # Вибір алгоритму
    print('Оберіть алгоритм для object tracking:')
    for i in range(len(algs)):
        print(i + 1, '-', algs[i])
    alg_mode = int(input('mode:'))
    # Якщо джерело даних існує
    if alg_mode in range(1, len(algs) + 1):

        # Зчитування відео
        cap = cv2.VideoCapture('{0}.mp4'.format(videos[data_mode - 1]))
        # Визначення параметрів запису відео
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Використовуйте 'mp4v', або 'XVID' для
кодування у mp4
        output_video = cv2.VideoWriter('{0}_{1}.mp4'.format(videos[data_mode - 1],
algs[alg_mode - 1]), fourcc, 60, (int(cap.get(3)), int(cap.get(4))))

        # Зчитування першого кадру
        ret, frame = cap.read()
        # Встановлення ROI (Region of Interest) - об'єкт, який відслідковуємо
        x, y, w, h = cv2.selectROI(frame)

        # MOSSE
        if (alg_mode == 1):
            tracker = cv2.legacy.TrackerMOSSE_create()

        # Ініціалізація трекера
        tracker.init(frame, (x, y, w, h))

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # Оновлення трекера
            ret, track_window = tracker.update(frame)
            # Малювання рамки відстежуваного об'єкту на кадрі
            x, y, w, h = int(track_window[0]), int(track_window[1]),
int(track_window[2]), int(track_window[3])
            img2 = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            # Запис кадра до вихідного відеофайлу
            output_video.write(frame)
            # Відображення результуючої рамки
            cv2.imshow('frame', img2)
            # Вихід з відео якщо натиснути 'q'
            if cv2.waitKey(1) == ord('q'):
                break

        cap.release()
        cv2.destroyAllWindows()
```

Результат:

Розглянемо результат на 3-х типових для вулиці відео **V_1.mp4, V_4.mp4, V_6.mp4**.

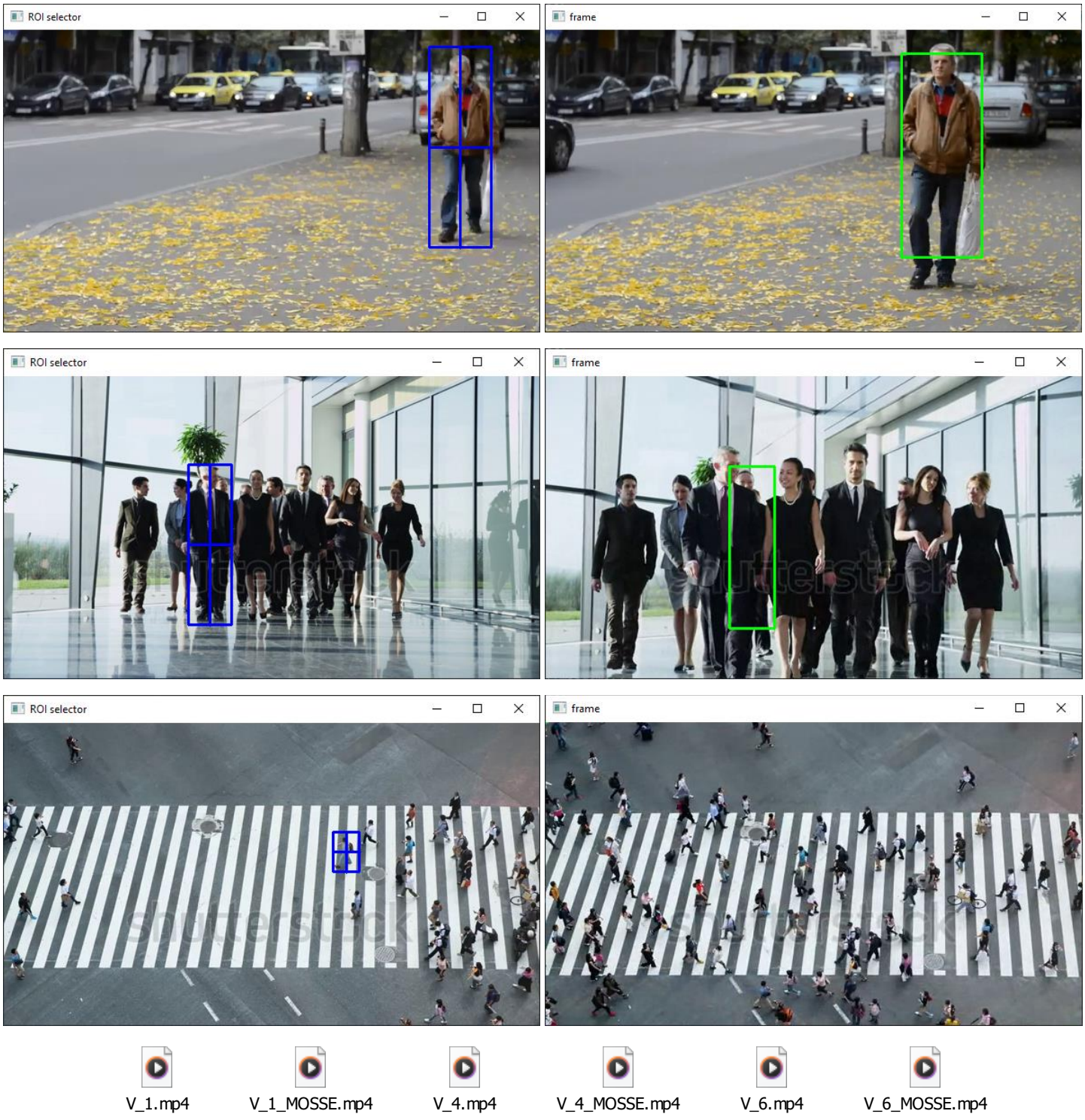


Рисунок 20 – Відстеження об’єктів алгоритмом **MOSSE**

Переглянувши відео-результати, можна побачити, що алгоритм відстежив обраного чоловіка на першому відео, але потім переключився на іншу людину. Це не добре для нашої задачі.

На другому відео йому важко відстежувати обрану людину і він збивається.

А на третьому відео він не зміг розпізнати обрану людину.

Алгоритм KCF для відстеження об'єктів

KCF (Kernelized Correlation Filters) — це сучасний алгоритм відстеження об'єктів, який використовує ядроподібний кореляційний фільтр для відстеження об'єкта. *KCF* дуже швидкий і точний і може обробляти зміни масштабу та обертання.

Лістинг коду:

```
[...]

# Вибір відео
[...]
# Якщо джерело даних існує
if data_mode in range(1, len(videos) + 1):

    # Вибір алгоритму
    [...]
    # Якщо джерело даних існує
    if alg_mode in range(1, len(algs) + 1):

        # Зчитування відео
        [...]
        # Визначення параметрів запису відео
        [...]
        # Зчитування першого кадру
        [...]
        # Встановлення ROI (Region of Interest) - об'єкт, який відслідковуємо
        [...]

        # KCF
        elif (alg_mode == 2):
            tracker = cv2.TrackerKCF_create()

        # Ініціалізація трекера
        [...]
            # Оновлення трекера
            [...]
            # Малювання рамки відстежуваного об'єкту на кадрі
            [...]
            # Запис кадра до вихідного відеофайлу
            [...]
            # Відображення результуючої рамки
            [...]
```

```
        # Вихід з відео якщо натиснути 'q'
        [...]
```

Результат:

Розглянемо результат на відео **V_1.mp4, V_4.mp4, V_6.mp4**.

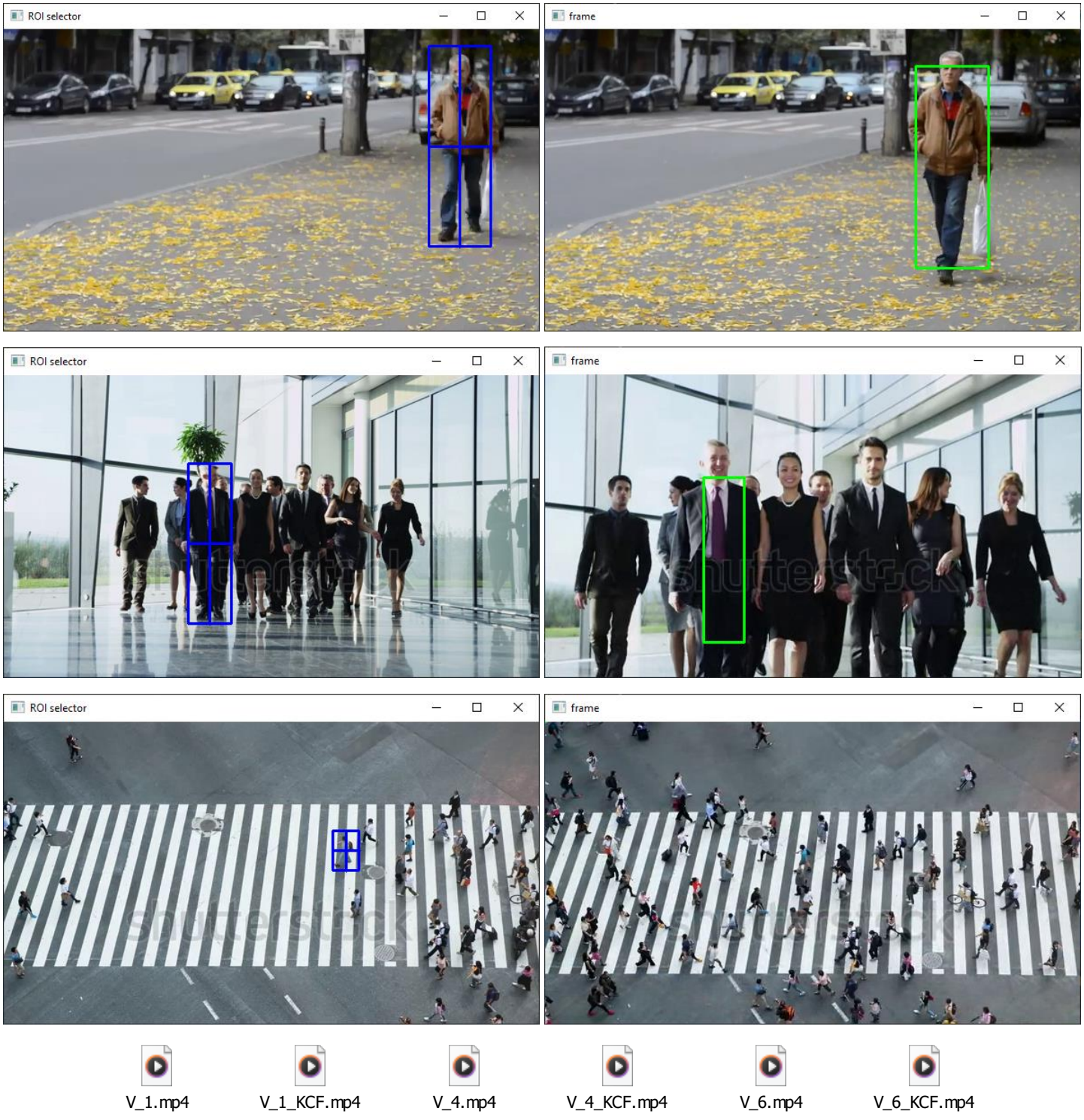


Рисунок 20 – Відстеження об’єктів алгоритмом **KCF**

Переглянувши відео-результати, можна побачити, що алгоритм чудово відстежив обраного чоловіка на першому відео потім рамка з відео зникла.

На другому відео алгоритм наплутав з розмірами рамки, але успішно відслідкував обраний об’єкт.

А на третьому відео він зміг розпізнати обрану людину лише на долю секунди.

Алгоритм CSRT для відстеження об'єктів

Цей алгоритм використовує карти просторової надійності для підстроювання опори фільтра до частини обраної області з кадру для відстеження, що дає можливість збільшувати зону пошуку та відстежувати непрямокутні об'єкти. Індекси надійності відображають якість досліджуваних фільтрів по каналах і використовуються як ваги для локалізації.

Лістинг коду:

```
[...]
# Вибір відео
[...]
# Якщо джерело даних існує
if data_mode in range(1, len(videos) + 1):
    # Вибір алгоритму
    [...]
    # Якщо джерело даних існує
    if alg_mode in range(1, len(algs) + 1):

        # Зчитування відео
        [...]
        # Визначення параметрів запису відео
        [...]
        # Зчитування першого кадру
        [...]
        # Встановлення ROI (Region of Interest) - об'єкт, який відслідковуємо
        [...]
        # CSRT
        elif (alg_mode == 3):
            tracker = cv2.TrackerCSRT_create()
        # Ініціалізація трекера
        [...]
            # Оновлення трекера
            [...]
            # Малювання рамки відстежуваного об'єкту на кадрі
            [...]
            # Запис кадра до вихідного відеофайлу
            [...]
            # Відображення результуючої рамки
            [...]
```

```
            # Вихід з відео якщо натиснути 'q'
            [...]
```

Результат:

Розглянемо результат на відео **V_1.mp4, V_4.mp4, V_6.mp4**.

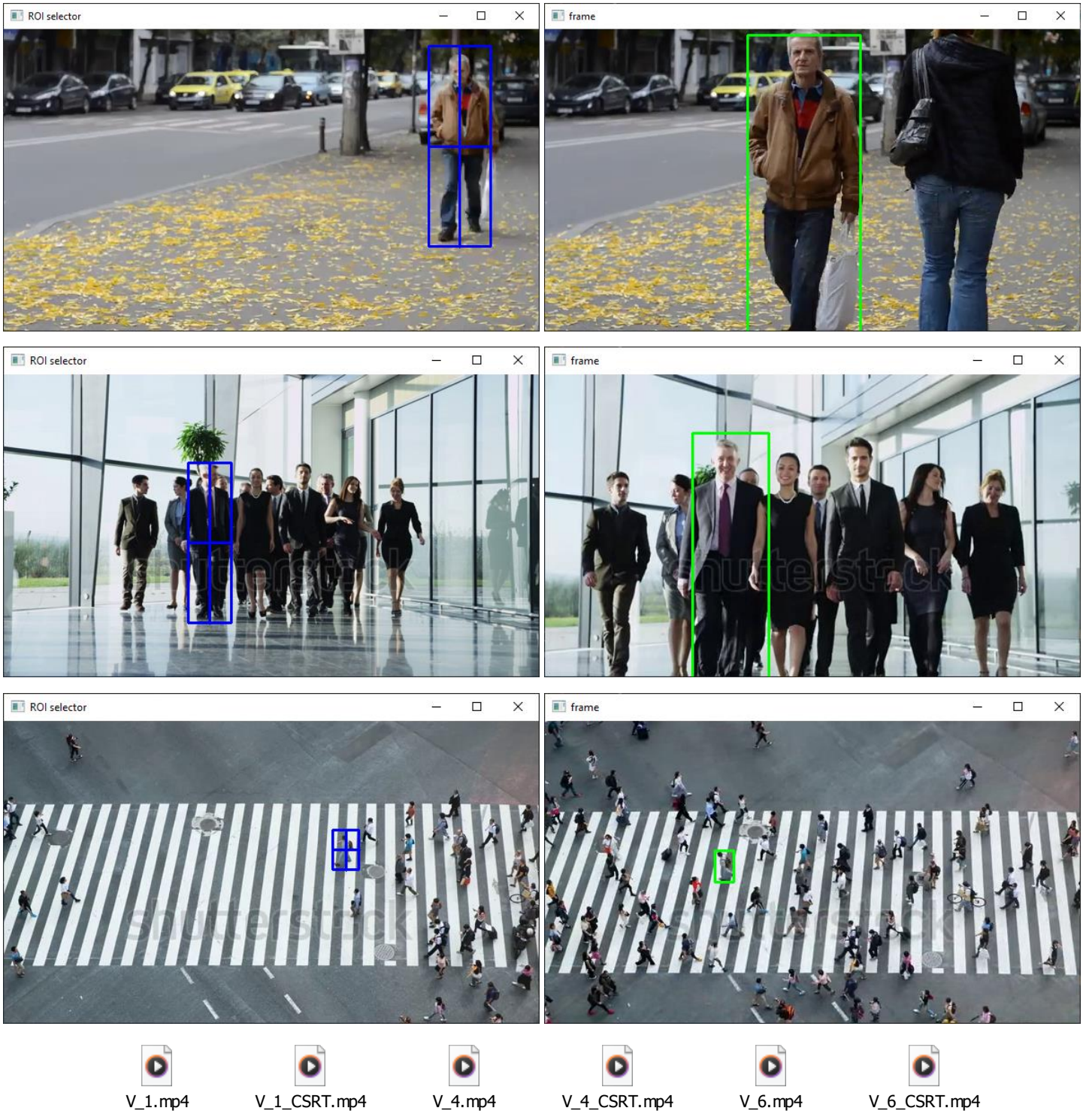


Рисунок 20 – Відстеження об’єктів алгоритмом **CSRT**

Переглянувши відео-результати, можна побачити, що алгоритм чудово відстежив обраного чоловіка на першому відео, але потім як у випадку *MOSSE* переключився на іншу людину.

На другому відео алгоритм чудово відслідкував обраний людину.

І нарешті на третьому відео він зміг розпізнати та від слідкувати до самого кінця обрану людину в натовпі. Єдине знову потім переключився на інший об’єкт.

2.4. Аналіз отриманих результатів

Сформульовано прикладні задачі, побудовано математичні моделі та розроблено програмний скрипт для трьох технічних завдань з областей *image recognition, image description, object tracking*.

Технічне завдання ***image recognition***:

Для задачі розпізнавання новорічних прикрас у вигляді куль розроблено програму, яка виявляє круглі об'єкти на цифровому зображенні, візуалізовує результат та підраховує кількість розпізнаних об’єктів.

Програма використовує **алгоритм Хаффа** для розпізнавання об’єктів.

Працездатність програми підтверджують результати тестування.

Рисунок 21 – Результати розпізнавання кіл

Отримані результати свідчать про те, що для успішного розпізнавання кіл на зображенні необхідно налаштовувати коефіцієнти алгоритму Хаффа під кожне окреме зображення. Алгоритму важче розпізнавати кола надто малого радіусу та багато кіл різного діаметру. На точність розпізнавання також впливає однотонність контура кола. Якщо коло затемнене знизу чи має відблиск, алгоритм розпізнає його з меншою ймовірністю.

Технічне завдання ***image description***:

Для задачі пошуку спільних рис у зруйнованих будівлях Маріуполя розроблено програму, яка знаходить ознаки у двох зображеннях і порівнює їх на відповідність.

Програма використовує **алгоритм Харріса** для розпізнавання кутів будівль та **дескриптор SIFT** для виділення спільних ознак у будівль з різного ракурсу.

Працездатність програми підтверджують результати тестування наведені на рисунках.

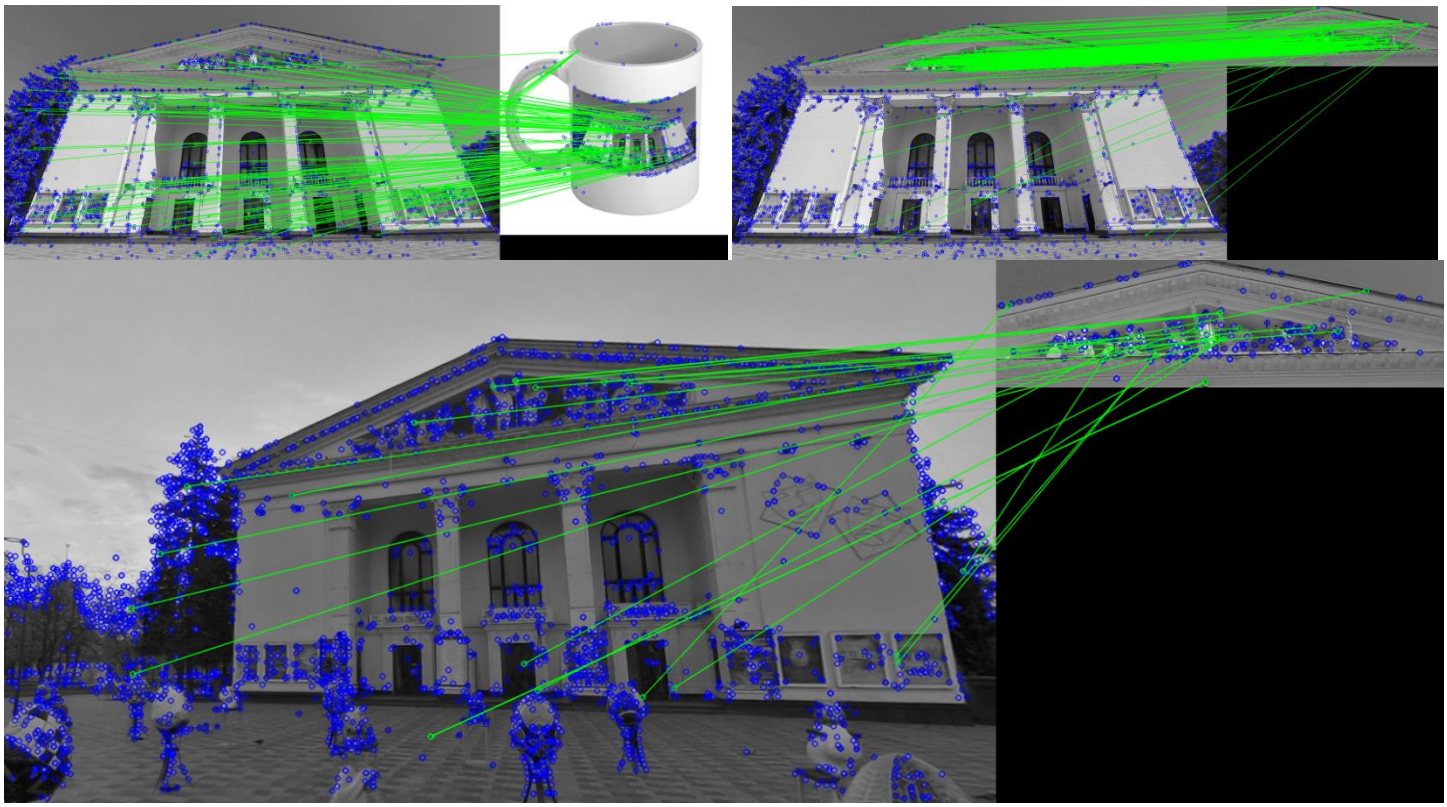


Рисунок 22 – Результати порівняння ознак двох зображень

Отримані результати свідчать про те, що алгоритм чудово розпізнає кути на зображеннях з високою контрастністю. Натомість на бляклих зображеннях він розпізнає менше ознак. Деформація у просторі та розмірі об’єкту не заважає розпізнавати у зображеннях спільні риси. Утім порівняння зображень з різного ракурсу потребує додаткових покращень програми.

Технічне завдання ***object detection***:

Для задачі відстеженя зловмисників по камерах відеоспостереження розроблено програму, яка відслідковує обраний об’єкт у відеопотоці.

Програма використовує **алгоритми MOSSE, KCF** та **CSRT** для розпізнавання та відстеження об’єкту.

Працездатність програми підтверджують результати тестування, збережені у вигляді відеофайлів у папці проекту. Результати наведені у файлах з іменем ***«назва початкового відео_алгоритм.mp4»***. Нижче наведені деякі скріншоти з відео-результатів.

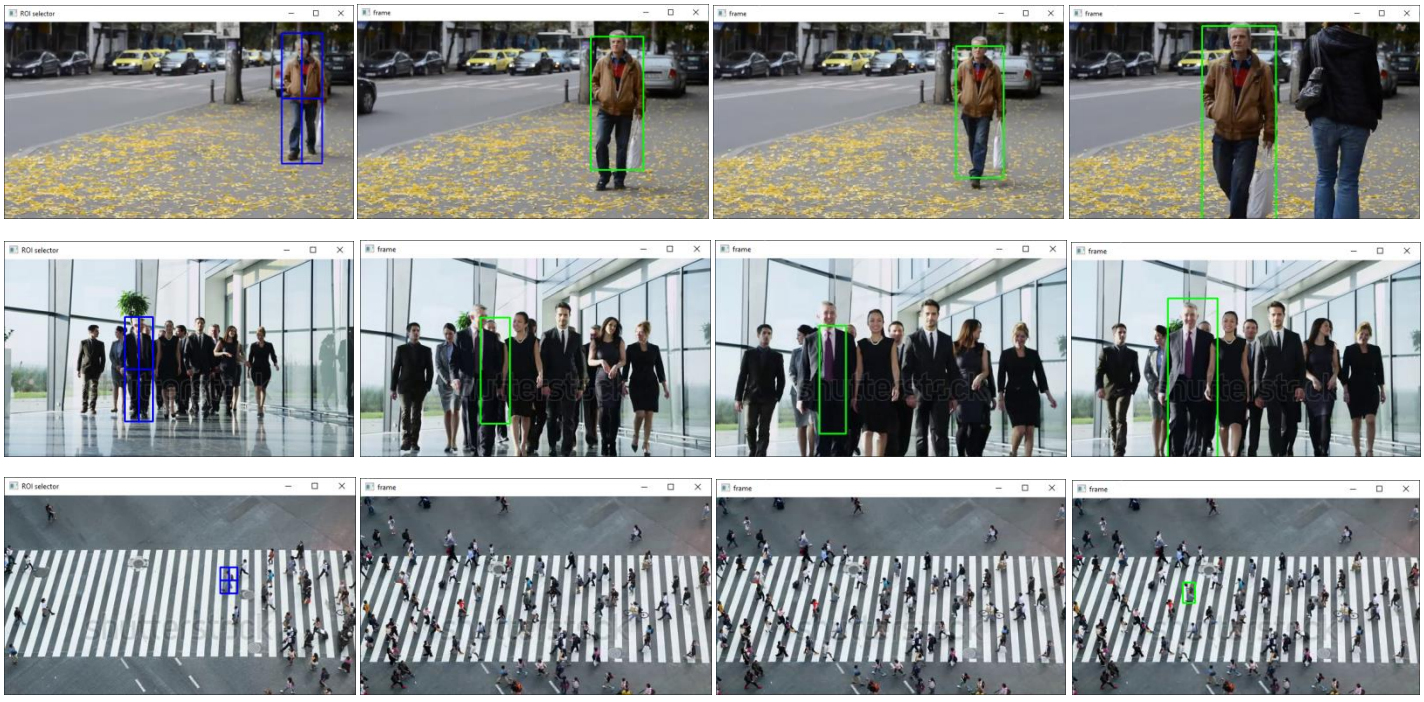


Рисунок 23 – Результати відстеження об’єкту у відеопотоці

Отримані результати свідчать про те, що усі розглянуті алгоритми мають свої особливості застосування. Алгоритм **MOSSE** найменш ресурсозатратний, але по якості

програє у всіх розглянутих випадках. Алгоритм **KCF** найкраще адаптується до відеорядів зі зніною розміру об’єкта. Коли об’єкт наближається або віддаляється від камери, алгоритм правильно змінює розмір рамки і продовжує слідкувати за обраним об’єктом. Алгоритм **CSRT** найкраще розпізнає та відстежує малі об’єкти та об’єкти на не дуже вираженому фоні (у натовпі).

Висновок:

Під час виконання лабораторної роботи розглянуто 3 області *Machine Learning*: **image recognition, image description** та **object detecting**.

Для групи технічних вимог 3 – застосовано **алгоритм Хаффа** для обробки для підрахунку та ідентифікації круглих об’єктів на зображенні. Результати вказують на високу точність обробки та ідентифікації об’єктів.

Для групи технічних вимог 4 – застосовано метод порівняння двох зображень з використанням **детектора кутів Харріса** та **дескриптора SIFT**. Отримані результати дозволяють оцінити ступінь схожості між зображеннями.

Для групи технічних вимог 5 – застосовано такі методи для ідентифікації об’єктів в відео потоці, як алгоритм **MOSSE, KCF** та **CSRT**. Розглянуті алгоритми забезпечують ефективне відслідковування об’єктів у реальному часі.

Здійснено тестування кожного з трьох програмних продуктів, результати якого наведено в звіті. Розроблені програми можуть бути використані у сформульованих у прикладних задачах сферах.