Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

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 ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

Мета: дослідити принципи та особливості практичного застосування технологій порівняння цифрових зображень для стеження за об'єктами у відеопотоці з використанням спеціалізованих програмних бібліотек.

Завдання:

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:

1) Дескриптор зображення має стосуватись об'єкта ідентифікації.
2) Алгоритм та технологія визначення дескриптора зображень – за власним вибором.
3) Результат порівняння – кількість збігів особливих точок;
4) Кількість збігів особливих точок дескриптора зображень перевести в ймовірність ідентифікації.

7-12	1. Оперативні: https://livingatlas2.arcgis.com/landsatexplorer/ 2. Високоточні: https://www.bing.com/maps	Район спостереження – обрати самостійно. Об'єкти ідентифікації – обрати самостійно. Дата оперативних даних – обрати самостійно. Метод і технологія сегментації / кластеризації – повинні забезпечувати можливість розрізнення та ідентифікацію обраних об'єктів спостереження.

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.

Синтезована математична модель:

Кути Харріса

Основна ідея цього методу - знайти області, де зміна інтенсивності є значною у всіх напрямках, що відповідає кутовим точкам.

Для кожного пікселя оцінюється зміна інтенсивності при невеликому зсуві (u,v) у зображенні. Ця зміна визначається функцією:

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

I(x,y) - інтенсивність пікселя у точці (x,y).

w(x,y) - функція вагового вікна (наприклад, гауссіан), що обмежує обчислення локальною областю.

(u,v) - зсув у горизонтальному (u) та вертикальному (v) напрямках.

E(u,v) - зміна інтенсивності при зсуві.

Для ефективності обчислень інтенсивність I(x+u,y+v) апроксимується за допомогою розкладу в ряд Тейлора до першого порядку:

$I(x+u,y+v) \approx I(x,y) + u \cdot I_x(x,y) + v \cdot I_y(x,y)$

де:

I_x і I_y - похідні інтенсивності по x та y.

Підставляючи це в E(u,v), отримуємо:

$$E(u,v) = \sum_{x,y} w(x,y) * \left[u * I_x(x,y) + v \cdot I_y(x,y)\right]^2$$

Це можна записати у матричній формі:

$$M = \sum_{x,y} w(x,y) * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x \cdot I_y & I_y^2 \end{bmatrix}$$

де:

I_x^2 та I_y^2 - квадрат градієнтів інтенсивності.

I_x та I_y - змішаний градієнт.

Для визначення, чи є піксель "кутом", розраховується відповідь функції Харріса:

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

det(M) = λ1 * λ2 — визначник матриці M (добуток власних значень).

trace(M) = λ1 + λ2 — слід матриці M (сума власних значень).

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

Загалом алгоритм Харріса забезпечує обчислення ключових точок, стійких до обертання, і є основою для багатьох сучасних алгоритмів виявлення особливостей, наприклад, SIFT чи SURF.

Результат архітектурного проєктування:

Порівняння об’єкта ідентифікації на декількох зображеннях:

Стеження за об’єктом у відеопотоці:

Опис структури програми:

*У папці **level1** містяться матеріали до завдання з 1 групи вимог:*

descriptor-water-detect.py – скрипт, що забезпечує виконання завдання

landsat_picture.png – зображення з операційного джерела

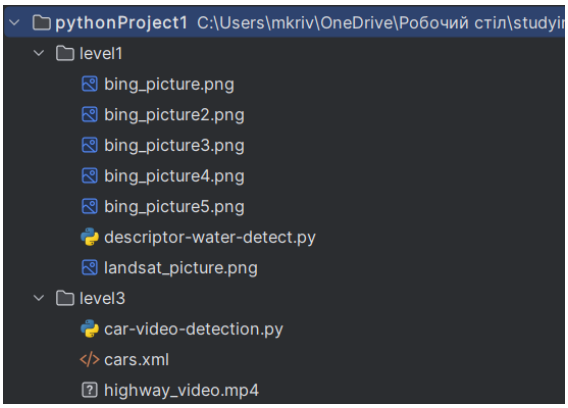
Все інше у цій папці – зображення із високоточного джерела з різних ракурсів

*У папці **level3** містяться матеріали до завдання з 3 групи вимог:*

car-video-detection.py - скрипт, що забезпечує виконання завдання

cars.xml – файл, що використовується для застосування каскадів Хаара

highway_video.mp4 – відео, на якому детектувались машини

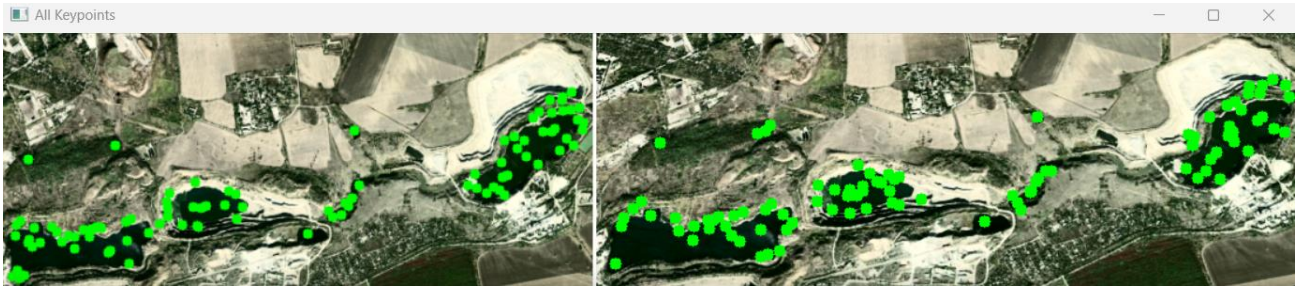


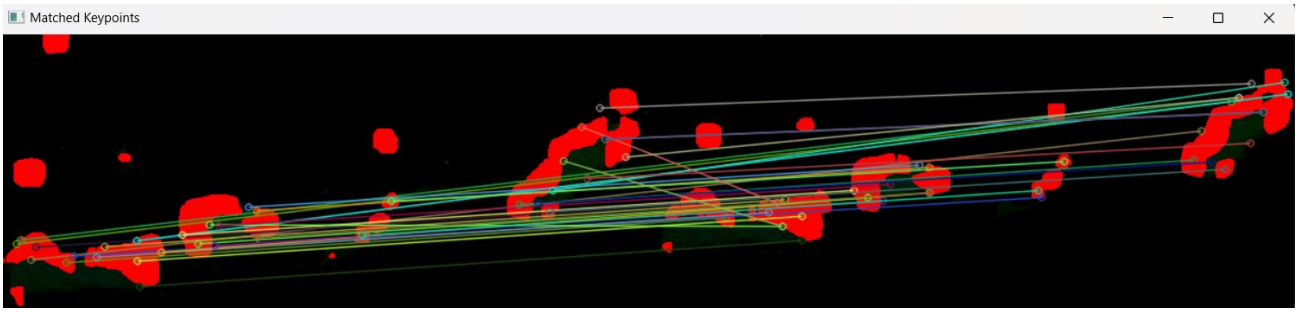
Результат роботи програми:

Порівняння об'єкта ідентифікації на декількох зображеннях:

bing_picture.png VS bing_picture2.png

Обравши два майже ідентичних зображення (друге зображення зроблене трохи під іншим кутом), можемо побачити, що більшість особливих точок правильно співставляються між двома зображеннями. Але з відсоткового результату ймовірності ідентифікації ми маємо зовсім невелике значення – всього 26.77%.

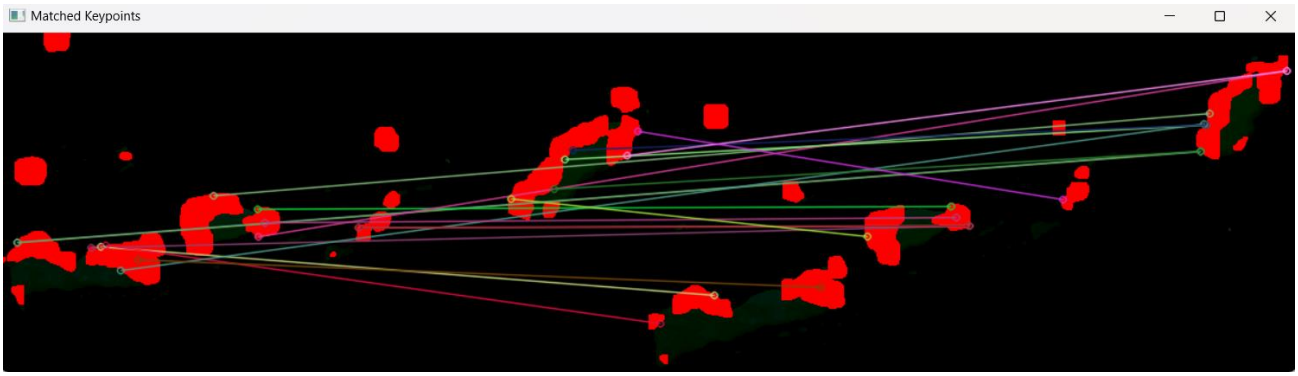





```
Number of matches: 34
Probability of identification: 26.77%
```

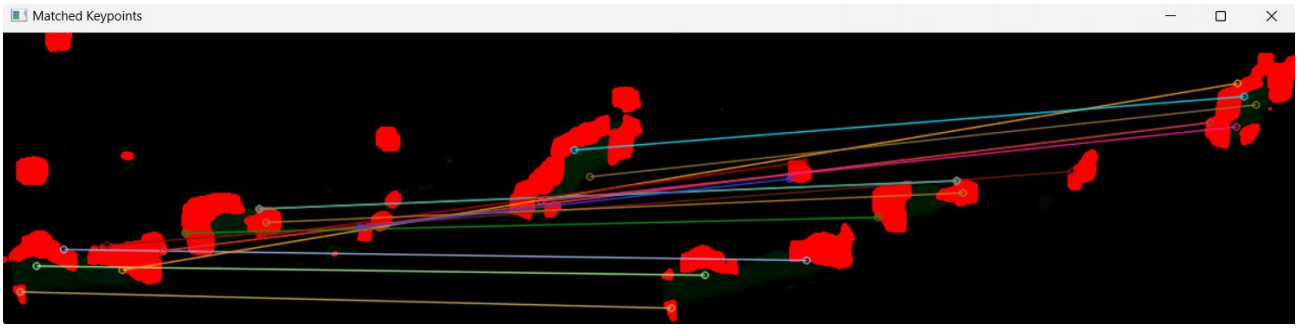
Спробуємо провести ще декілька експериментів над зображеннями з різними ракурсами і подивимось, які відсоткові значення ми зможемо отримати:

bing_picture.png VS bing_picture3.png




```
Number of matches: 19
Probability of identification: 17.59%
```

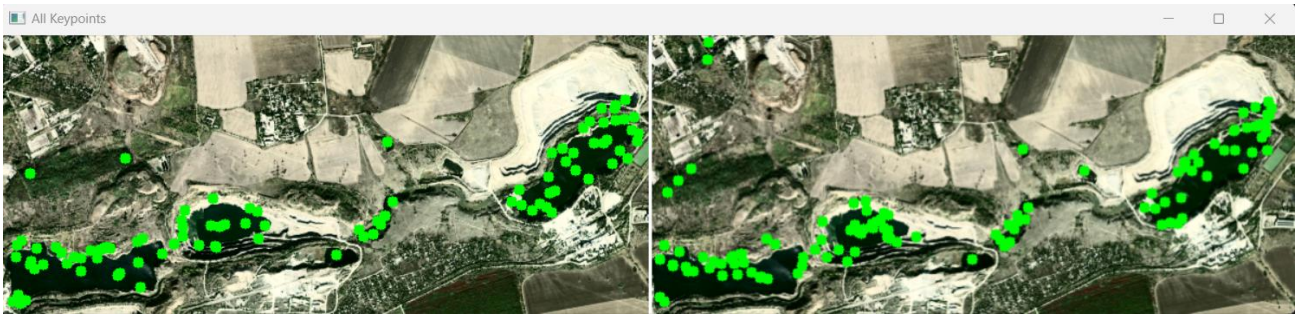
bing_picture.png VS bing_picture4.png

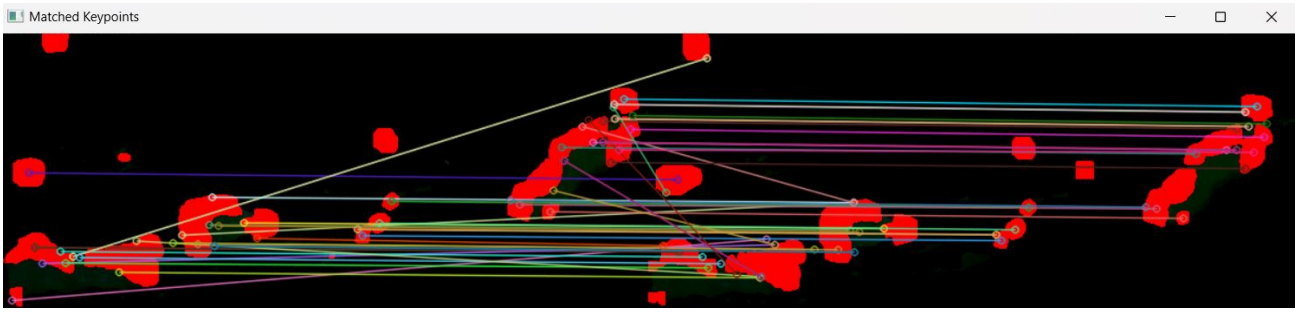


```
Number of matches: 16
Probability of identification: 15.69%
```

bing_picture.png VS bing_picture5.png

У цьому вже видно значне збільшення точок, що збіглись, але все одно воно не настільки велике, як ми могли очікувати. Таке збільшення відсотку дуже легко пояснюється: *цього разу ми не змінювали нахил і поворот в ракурсі зображення, а просто змістили його трохи вбік – тим самим ми не викривлювали кути об'єктів і дали програмі працювати якомога ефективніше.* Звісно, є декілька ключових точок, що з'єднані некоректно, але їх меншість.






```
Number of matches: 51
Probability of identification: 40.16%
```

Продивившись всі результуючі зображення зі з'єднаними ключовими точками, можемо впевнено сказати, що програма дійсно здебільшого правильно співвідносить їх.

Але як же тоді бути із малими значеннями відсотків? Враховуючи те, що в останньому порівнянні зображення були візуально майже ідентичні, там не було викривлення кутів через зміну кута в ракурсі і там було знайдено найбільше збігів серед ключів, то є сенс "відкалібрувати" раніше отримані результати.

Тоді вийде, що:

bing_picture.png VS bing_picture5.png **= 100% (раніше 40.16%)**

bing_picture.png VS bing_pictur2.png = **66.66% (раніше 26.77%)**

bing_picture.png VS bing_picture3.png = **43.79 (раніше 17.59%)**

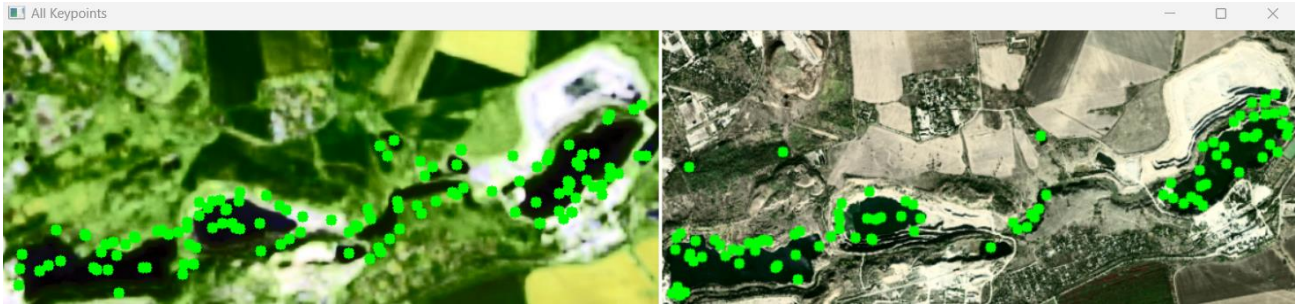
bing_picture.png VS bing_picture4.png = **39.07% (раніше 15.69%)**

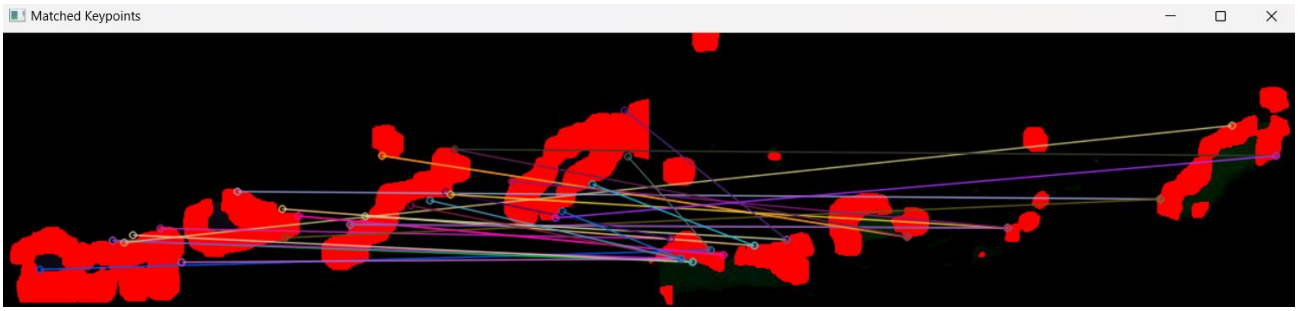
Із цих результатів можемо зробити висновок, що алгоритм не дуже чутливий до зміщення об’єкта ідентифікації, але при цьому чутливий до змінення його форми за рахунок зміни кута спостереження, бо таким чином викривлюються кути об’єкта ідентифікації. Найбільші відсотки схожості отримають ті пари зображень, що лише зміщують об’єкт ідентифікації у різні частини зображення.

Спробуємо також порівняти зображення із високоточного джерела та оперативного між собою:

landsat_picture.png VS bing_picture.png

Бачимо, що кількість збігів не така й мала, але через різницю “чутливості до деталей” цих двох зображень маємо не дуже коректно з’єднані ключові точки.



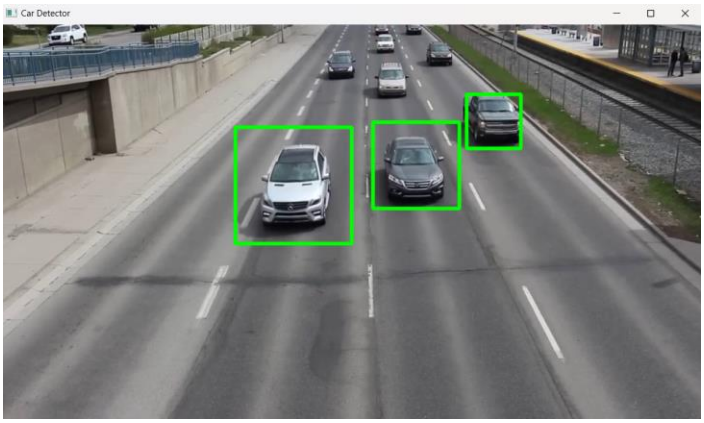


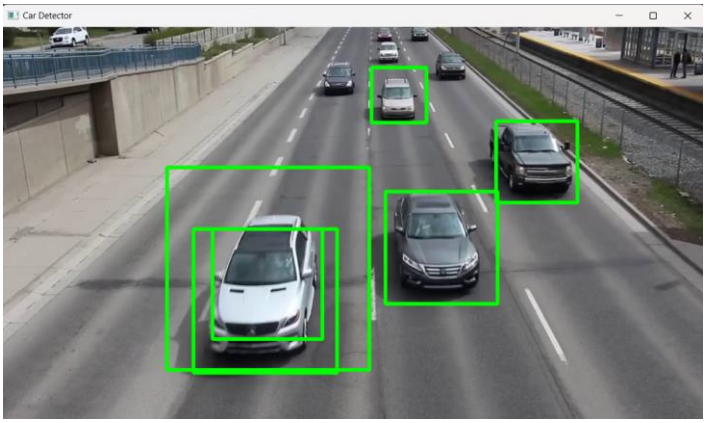

```
Number of matches: 29
Probability of identification: 22.83%
```

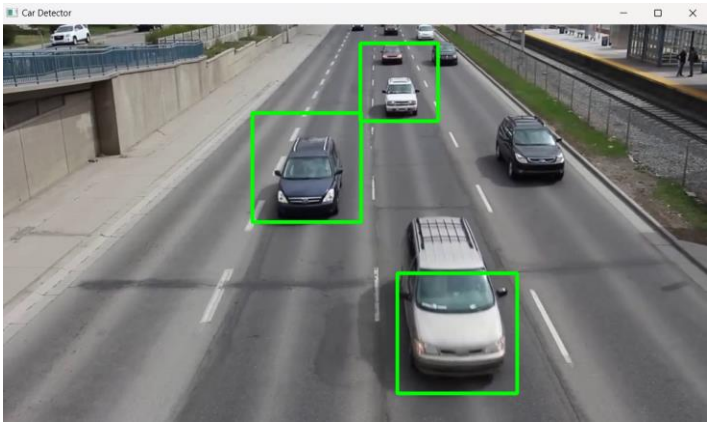
Стеження за об’єктом у відеопотоці:

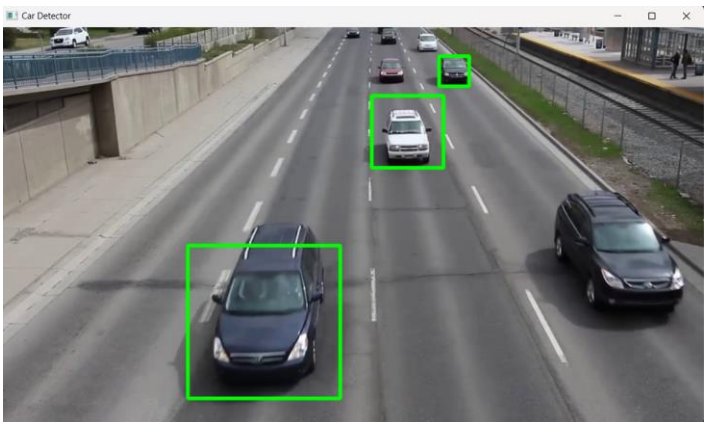
В рамках цього завдання було взято те саме відео, що і у 4 лабораторній роботі для порівняння різних способів ідентифікації об’єктів. Файл xml для застосування каскадів Хаара був взятий із відкритого репозиторія https://github.com/abhi-kumar/CAR-DETECTION/tree/master . Також було залишено маску, використану у 4 лр для того, щоб алгоритм аналізував на наявність машин виключно центральну дорогу.

Із наступних скріншотів можна легко побачити, що ця версія програми працює набагато стабільніше за мою реалізацію у 4 лабораторній і чіткіше визначає автомобілі. Це пояснюється застосуванням більш просунутого підходу до ідентифікації і відсутності сильної прив’язаності до кольору та підбору безлічі параметрів.


Car Detector


Car Detector


Car Detector


Car Detector

Також варто зазначити, що такий варіант набагато легший у реалізації ніж імплементація в 4 лр, де все дуже сильно залежало від декількох параметрів, підібраних внаслідок десятка експериментів.

Програмний код, що забезпечує отримання результату:

Порівняння об'єкта ідентифікації на декількох зображеннях:

```
import cv2
import numpy as np
import imutils

def color_correction(image_path):
    image = cv2.imread(image_path)
    image = imutils.resize(image, width=600)
    # Color correction - simple histogram equalization in LAB color space
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    l = cv2.equalizeHist(l)
    lab = cv2.merge((l, a, b))
    corrected_image = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
    return corrected_image

def color_clustering(image, lower_water, upper_water):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower_water, upper_water)
    img_bitwise_and = cv2.bitwise_and(image, image, mask=mask)
    # Median blurring
    img_median_blurred = cv2.medianBlur(img_bitwise_and, 5)
    return mask, img_median_blurred

def find_contours(image, mask):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    return contours

def identify_objects(image, contours):
    identified_image = image.copy()
    for i, contour in enumerate(contours):
        area = cv2.contourArea(contour)
        if area > 100:  # Filter small areas
            M = cv2.moments(contour)
            if M['m00'] != 0:
                cX = int(M['m10'] / M['m00'])
                cY = int(M['m01'] / M['m00'])
                cv2.circle(identified_image, (cX, cY), 5, (0, 255, 0), -1)  #
Draw circle at center
    return identified_image

def harris_corner_detector(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv2.cornerHarris(gray, 20, 9, 0.0000001)
    dst = cv2.dilate(dst, None)
    image[dst > 0.01 * dst.max()] = [0, 0, 255]  # Mark corners in red
    return image

def sift_descriptors_on_harris(image):
    # Create SIFT detector
```

```
    sift = cv2.SIFT_create()
    kp, des = sift.detectAndCompute(image, None)
    return kp, des

def sift_feature_matching(kp1, des1, kp2, des2):
    # Create a FLANN matcher
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    # Match descriptors
    matches = flann.knnMatch(des1, des2, k=2)

    # Filter matches using the Lowe's ratio test
    good_matches = []
    for m, n in matches:
        if m.distance < 0.7 * n.distance:
            good_matches.append(m)

    return good_matches

def draw_matches(img1, kp1, img2, kp2, matches):
    matched_image = cv2.drawMatches(img1, kp1, img2, kp2, matches, None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    return matched_image

def process_images(operational_image_path, high_res_image_path):
    # Color correction
    operational_image = color_correction(operational_image_path)
    high_res_image = color_correction(high_res_image_path)

    # For two different bing pictures
    # bing_picture.png
    op_lower_water = np.array([55, 150, 11])
    op_upper_water = np.array([100, 255, 255])
    # bing_picture2.png
    hg_lower_water = np.array([55, 150, 11])
    hg_upper_water = np.array([100, 255, 255])

    # For landsat_picture and bing_picture
    # landsat_picture.png
    # op_lower_water = np.array([25, 0, 0])
    # op_upper_water = np.array([240, 255, 30])
    # bing_picture.png
    # hg_lower_water = np.array([55, 150, 11])
    # hg_upper_water = np.array([100, 255, 255])

    # Color clustering
    operational_mask, op_img_median_blurred =
color_clustering(operational_image, op_lower_water, op_upper_water)
    high_res_mask, hg_img_median_blurred = color_clustering(high_res_image,
hg_lower_water, hg_upper_water)

    # Find contours
    operational_contours = find_contours(operational_image, operational_mask)
    high_res_contours = find_contours(high_res_image, high_res_mask)

    # Identifying objects
```

```
    operational_ident_image = identify_objects(operational_image,
operational_contours)
    high_res_ident_image = identify_objects(high_res_image, high_res_contours)

    # Apply Harris corner detection
    operational_harris_image = harris_corner_detector(op_img_median_blurred)
    high_res_harris_image = harris_corner_detector(hg_img_median_blurred)

    # Get SIFT descriptors based on Harris corners
    kp1, des1 = sift_descriptors_on_harris(operational_harris_image)
    kp2, des2 = sift_descriptors_on_harris(high_res_harris_image)

    # Perform SIFT feature matching
    good_matches = sift_feature_matching(kp1, des1, kp2, des2)

    # Draw keypoints on the images
    for kp in kp1:
        cv2.circle(operational_ident_image, (int(kp.pt[0]), int(kp.pt[1])), 5,
(0, 255, 0), -1)  # Green circles
    for kp in kp2:
        cv2.circle(high_res_ident_image, (int(kp.pt[0]), int(kp.pt[1])), 5, (0,
255, 0), -1)  # Green circles

    # Resize images to have the same height for side-by-side display
    height1, width1 = operational_ident_image.shape[:2]
    height2, width2 = high_res_ident_image.shape[:2]
    new_height = min(height1, height2)

    operational_ident_resized = imutils.resize(operational_ident_image,
height=new_height)
    high_res_ident_resized = imutils.resize(high_res_ident_image,
height=new_height)

    # Create side by side display
    combined_image = np.hstack((operational_ident_resized,
high_res_ident_resized))

    # Draw matches
    if good_matches:
        matched_image = draw_matches(op_img_median_blurred, kp1,
hg_img_median_blurred, kp2, good_matches)
    else:
        matched_image = np.zeros((600, 1200, 3), dtype=np.uint8)  # Create an
empty image if no matches found

    # Draw a white line in the middle to separate the two images
    line_position = operational_ident_resized.shape[1]  # Position of the line
    cv2.line(combined_image, (line_position, 0), (line_position, new_height),
(255, 255, 255), 2)

    # Calculate probability of identification
    total_descriptors = min(len(kp1), len(kp2))
    match_count = len(good_matches)
    probability = (match_count / total_descriptors) * 100 if total_descriptors >
0 else 0

    print(f'Number of matches: {match_count}')
    print(f'Probability of identification: {probability:.2f}%')
```

```
    # Show results
    cv2.imshow("All Keypoints", combined_image)
    cv2.imshow("Matched Keypoints", matched_image)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    operational_image_path = 'bing_picture.png'
    high_res_image_path = 'bing_picture4.png'
    process_images(operational_image_path, high_res_image_path)
```

Стеження за об'єктом у відеопотоці:

```
import time

import cv2
import numpy as np

def create_road_mask(img):
    height, width = img.shape[:2]
    road_mask = np.zeros((height, width), dtype=np.uint8)

    pts = np.array([
        [0, int(height * 0.8)],                  # Near bottom-left corner
        [0, height],                             # Bottom-left corner
        [width, height],                         # Bottom-right corner
        [width, int(height * 0.6)],              # Near bottom-right corner
        [int(width * 0.6), int(height * 0.01)],  # Near top-right
        [int(width * 0.5), int(height * 0.01)]   # Near top-left
    ], np.int32)

    # Filling the mask with the trapezoidal region
    cv2.fillPoly(road_mask, [pts], 255)

    masked_img = cv2.bitwise_and(img, img, mask=road_mask)

    return masked_img, road_mask


if __name__ == "__main__":
    # Load the Haar cascade file
    car_cascade = cv2.CascadeClassifier('cars.xml')

    # Check if the cascade file has been loaded correctly
    if car_cascade.empty():
        raise IOError('Unable to load the car cascade classifier xml file')

    cap = cv2.VideoCapture('highway_video.mp4')

    # Define the scaling factor
    scaling_factor = 0.5

    # Iterate until the user hits the 'Esc' key
    while True:
        time.sleep(0.03)
        # Capture the current frame
        ret, frame = cap.read()
```

```
        if not ret:
            break  # Exit if no more frames

        # Resize the frame
        frame = cv2.resize(frame, None,
                           fx=scaling_factor, fy=scaling_factor,
                           interpolation=cv2.INTER_AREA)

        # Create a road mask for the current frame
        masked_img, road_mask = create_road_mask(frame)

        # Convert the masked image to grayscale
        gray = cv2.cvtColor(masked_img, cv2.COLOR_BGR2GRAY)

        # Run the car detector on the grayscale image
        car_rects = car_cascade.detectMultiScale(gray, 1.3, 5)

        # Draw a rectangle around the detected cars
        for (x, y, w, h) in car_rects:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

        cv2.imshow('Car Detector', frame)

        # Check if the user hit the 'Esc' key
        c = cv2.waitKey(1)
        if c == 27:  # Esc key
            break

    cap.release()

    # Close all the windows
    cv2.destroyAllWindows()
```

Висновки: виконавши цю лабораторну роботу, я навчилась визначати ключові точки зображень за допомогою кутів Харріса та порівнювати їх із ключовими точками іншого зображення. Завдяки цьому я змогла реалізувати програму, що порівнює розташування озер поблизу Запоріжжя на різних супутникових фото. Внаслідок експериментів із розробленим скриптом, було визначено, що він чутливий до кута спостереження на зображенні, але є значно менш чутливим до зміщення. Також я навчилась працювати із каскадами Хаара, завдяки чому змогла реалізувати скрипт, що ідентифікує машини на відеопотоці.