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 №4
з дисципліни
«Технології Computer Vision»
на тему
«Дослідження технологій покращення якості цифрових зображень для задач Computer Vision»

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 покращення якості цифрових зображень для задач Computer Vision з використанням спеціалізованих програмних бібліотек.

Завдання:

Здійснити R&D дослідження та реалізувати програмний скрипт із конкретикою методів і технологічних етапів Computer Vision: вибір цифрового зображення та об'єкта ідентифікації; завантаження цифрового зображення; покращення якості цифрового зображення; векторизація об'єкта ідентифікації – встановлення геометричної ознаки; ідентифікація об'єкта за геометричною ознакою.

Вибір цифрового зображення та об'єкту ідентифікації встановлено варіантами таблиці додатку.

7	Автомагістраль	Легкові автомобілі

Для покращення якості цифрового зображення використовувати:корекцію кольору; корекцію гістограми яскравості (для всього зображення (глобальна) / для сегменту зображення (локальна)); методи / алгоритми фільтрації зображень.

Вибір переліку методів покращення якості має бути обґрунтованим та забезпечувати побудову контору об'єкту ідетифікації.

Для векторизації зображення (визначення контуру) використовувати методи базових бібліотек python для обробки цифрових зображень;

Ідентифікацію здійснювати за технологією порівняння геометричних ознак (контуру) образу та об'єкту ідентифікації.

В рамках цієї роботи було **виконано завдання обидвох рівнів**:

Завдання І рівня – максимально 8 балів.

Здійснити виконання завдання лабораторної роботи для статичного цифрового зображення за варіантами таблиці додатку.

Завдання ІІ рівня – максимально 9 балів.

Здійснити виконання завдання лабораторної роботи для відеопотоку за варіантами таблиці додатку.

Результати виконання лабораторної роботи:

Синтезовані математичні моделі:

Робота з маскою

Задаємо вершини багатокутника, що покриває форму дороги

$Vertices\ V = \{(x_i, y_i) \mid i = 1, 2, \dots, 6\}$

Всі координати задавались із таким розрахунком, щоб маска покрила проїжджу частину.

Створюємо маску

$$M(x,y) = \begin{cases} 1 & \text{if}(x,y)\ \epsilon\ \text{Polygon(V)} \\ 0 \end{cases}$$

Накладаємо маску на зображення

$$\text{I}_{\text{masked}}(x,y)\ =\ \text{I}(x,y)\ *\ \text{M}(x,y)$$

<u>Робота з фільтром</u>

Спочатку створюємо дві нові кольорові маски C1(x, y) та C2(x, y) для граничних значень

$$\text{C}_1(x,y) = \begin{cases} 1 & \text{if}(\text{H},\text{S},\text{V})\ \epsilon\ \text{Range}_{\text{Light}} \\ 0 \end{cases}$$

$$\text{C}_2(x,y) = \begin{cases} 1 & \text{if}(\text{H},\text{S},\text{V})\ \epsilon\ \text{Range}_{\text{Dark}} \\ 0 \end{cases}$$

Комбінуємо маски між собою

$$\text{C}(x,y)\ =\ \text{C}_1(x,y)\ \cup\ \text{C}_2(x,y)$$

Створюємо адаптивний поріг

$$\text{T}(x,y) = \begin{cases} 1 & \text{if (min - pix}\ >\ \text{const)} \\ 0 \end{cases}$$

min – локальний мінімум

pix – значення пікселя

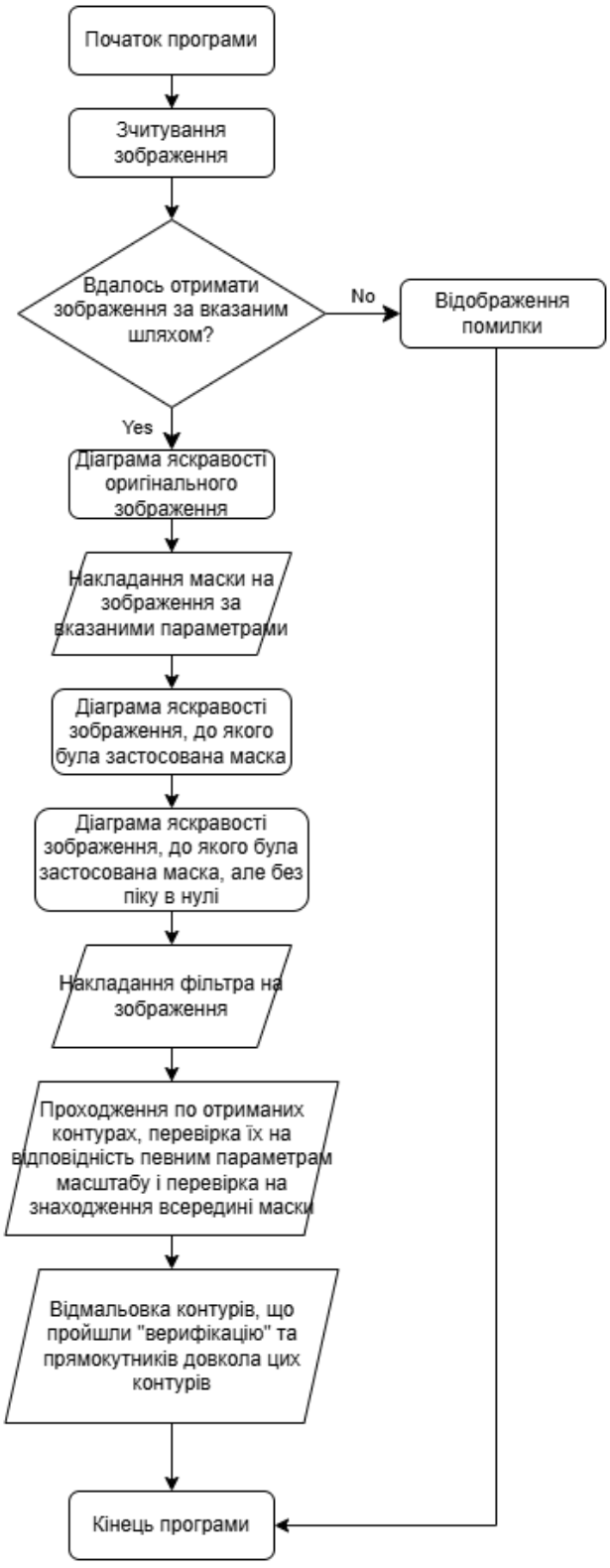
const – певна константа, що підбирається вручну шляхом експерименту

Далі вже формується результуюче зображення

```
kernel = np.ones((2, 2), np.uint8)
filtered = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
filtered = cv2.morphologyEx(filtered, cv2.MORPH_OPEN, kernel)
```

Результати архітектурного проектування:

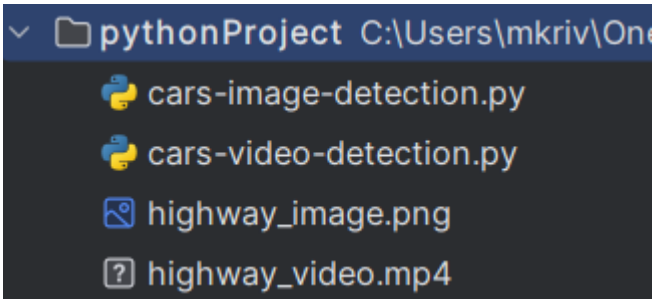
Блок-схема функціонування програми, що розпізнає автівки на статичному зображенні:



Програма, що розпізнає автівку на **відео**, має схожу структуру – для кожного кадру відео вона застосовує алгоритм розпізнавання, що був зазначений вище. Єдина відмінність – цього разу ніякі графіки не виводяться. Програма прокручує відео із розпізнаними машинами та окремо просто відфільтроване відео.

Опис структури проекту програми:

Структура проєкту дуже проста – файли із двома програмами: одна для розпізнавання машин на фото, інша – на відео. Саме зображення та відео також знаходяться в проєкті

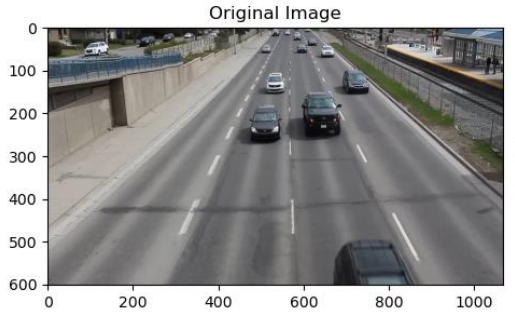


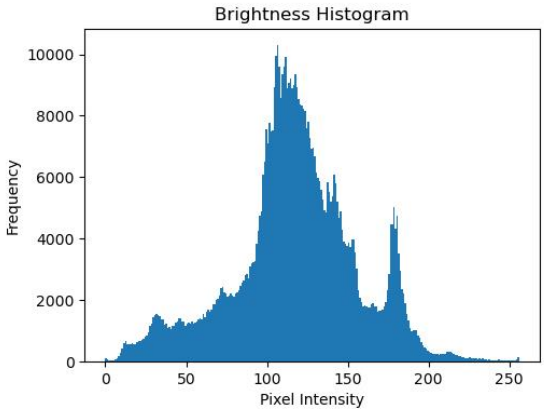
Результати роботи програми відповідно до завдання:

Статичне зображення:

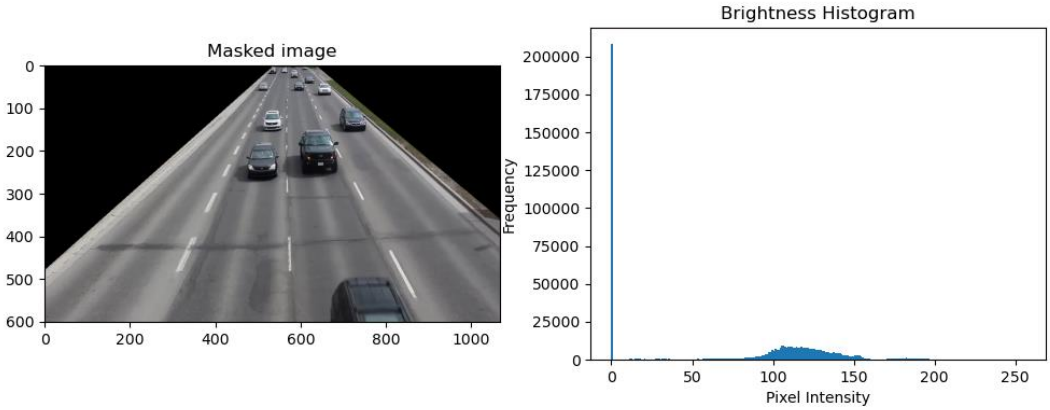
На поданих нижче скріншотах бачимо результат роботи програми на прикладі одного із зображень автомагістралі з машинами.

З першої гістограми яскравості складно дістати цінну інформацію, адже зображення далеко не однотонне та воно містить машини абсолютно різних кольорів. Тому без належної обробки та накладання маски ми не зрозуміємо справжній спектр дороги – об’єкта, що нам цікавий.





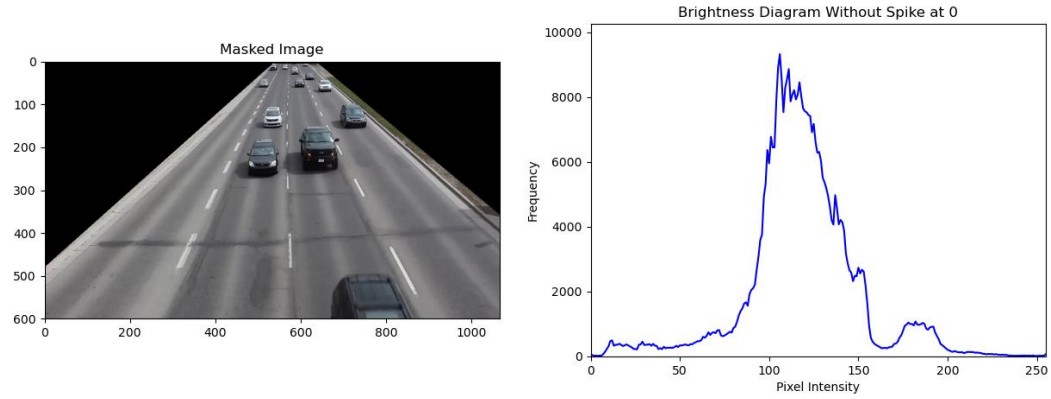
Після накладання маски для врахування виключно області дороги вже можемо побачити цікавішу картинку. Великий пік у 0 – чорний колір маски, а от все інше ілюструє дорогу.



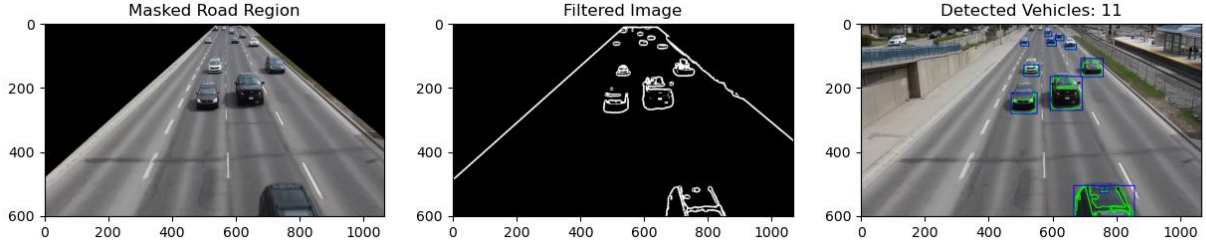
Спробуємо “приблизити” цю діаграму, відкинувши з неї пік в районі нулів, щоб детальніше розгледіти всі інші діапазони (тут зробила не гістограмою, щоб було зручніше дивитись).

Якщо порівняти із тим, що ми бачили на найпершій гістограмі, то вже чітко можна побачити як зменшилась кількість світлих пікселів. Це пояснюється тим, що велика частина зображення поза дорогою мала характерний світлий колір.

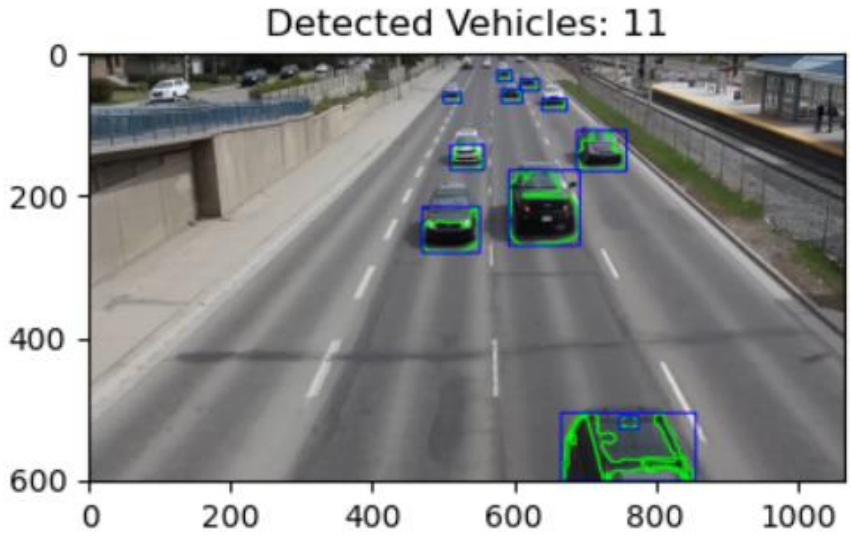
Дивлячись на цей графік, вже можна висувати припущення. Ми вже виявили, що близько 0 знаходяться дуже темні регіони. Тому частина графіка від 0 до ~50 ймовірніше за все відповідає за темні автівки та тіні. Найвищий пік скоріше за все відповідає за асфальтне покриття, адже його частка на зображенні найбільша. А пік від 150 до 200 ймовірно показує нам на світлі полоски різних відтінків на дорозі і на світлі машини.



Остаточний результат фільтрації та ідентифікації автівок можна побачити на наступному зображенні (вище було описано математичні моделі роботи із фільтром):



Бачимо, що підрахована кількість автомобілів на 1 більше, ніж кількість справжніх автомобілів. Якщо придивитись, то можна побачити, що у найближчій до нас машині окремо синьою рамкою виділено частину даху. Ця частинка була хибно розпізнана програмою як автівка через різницю яскравостей.

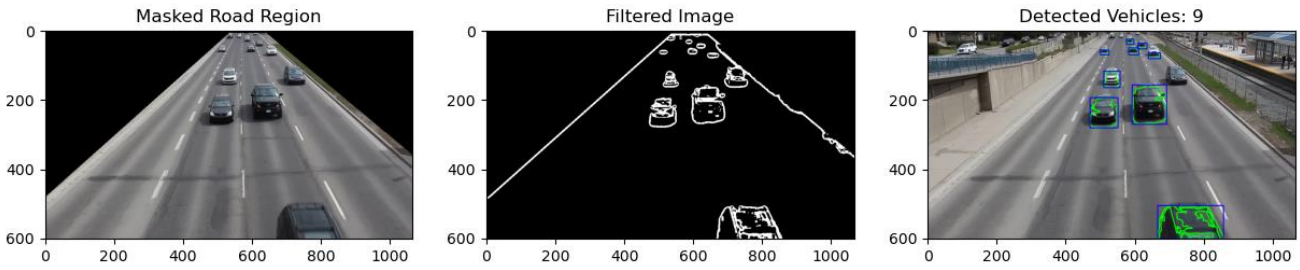


Таку проблему можна було б легко вирішити, відкоригувавши частину коду, що відповідає за мінімально допустимий розмір ідентифікованої машини, але в такому разі авто вдалині також не будуть розпізнаватись.

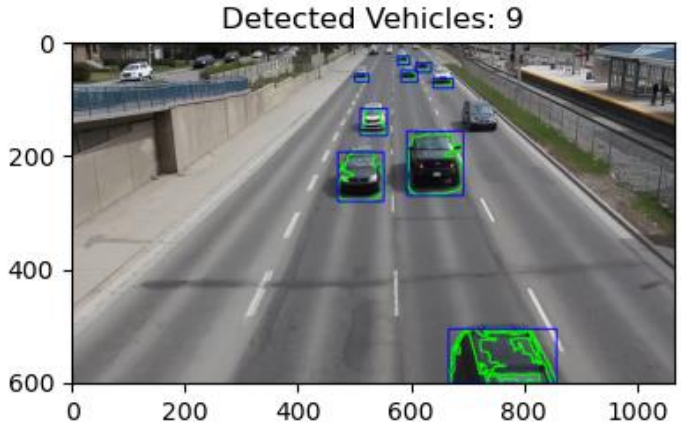
```
for contour in contours:
    area = cv2.contourArea(contour)
    if area < 200:
        continue
```

З метою більшої ефективності програми, я вирішила залишити наявні параметри і не жертвувати ідентифікацію машин вдалині.

Також наявні результати **проміжної версії програми із *іншими* параметрами** кольорової маски:



Окрему увагу варто звернути на фінальне зображення цієї версії програми, де було розпізнано не всі машини. Можемо помітити, що одна з машин у правій смузі була «проігнорована» програмою.



Чому ж так сталося? Якраз цей приклад і є показовою ілюстрацією того, що подібний метод фільтрації та визначення контуру – не завжди найефективніший. Декілька параметрів фільтрації та маска у програмі задавались вручну - це є стримуючим фактором, адже лімітує кількість комбінацій параметрів. Не всі комбінації працюють однаково добре, а отже необхідно шукати працюючий варіант майже повністю вручну.

Не виключено, що програма працювала не зовсім коректно через те, що контур машини пересікав контур маски. У попередньо показаній працюючій версії програми ця проблема вирішилась зміною параметрів одної з кольорових масок.

Тож, не зважаючи на те, що в результаті фільтрації ми очима досить чітко бачимо контури всіх машин, погрішності все ж можуть ставатись коли ми використовуємо підхід до ідентифікації об’єктів, який залежний від введених вручну змінних.

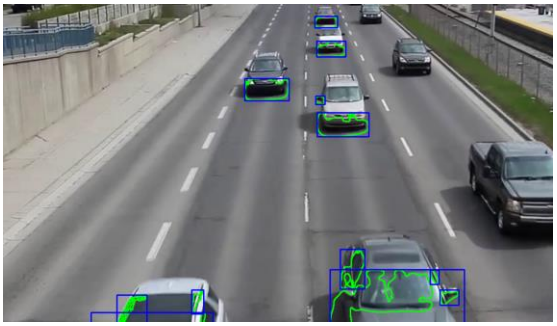
В даному випадку **більшість погрішностей вдалось виправити**, але такий позитивний сценарій можливий не завжди, і це потрібно враховувати.

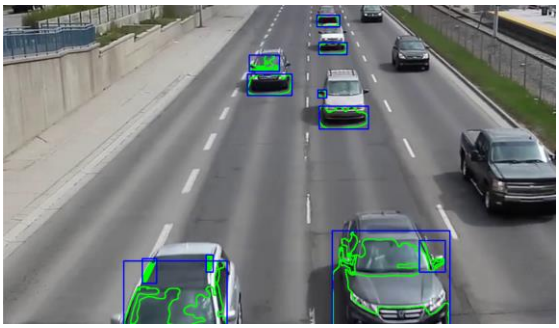
Відеопотік:

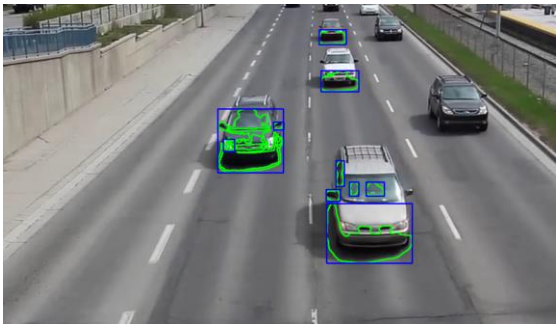
Так як всередині програма використовує той самий алгоритм, що і програма для розпізнавання машин на статичному зображенні, наведу просто декілька кадрів.

Дуже просто помітити, що автомобілі тут ідентифікуються менш точно ніж на статичному зображенні. Чому так, якщо використовується один алгоритм? Знову ж таки, причина у заданих вручну параметрах у певних частинах програми. Те, що працює добре для стоп-кадра, не обов’язково працюватиме так само для цілого відео. **Хоча й на відфільтрованому відео ми очима +- чітко бачимо обриси кожної машини.**

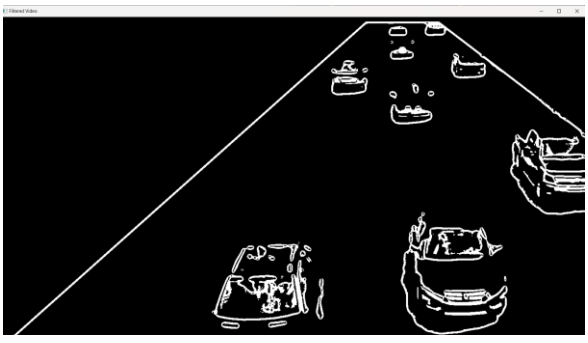
Оригінальне відео:

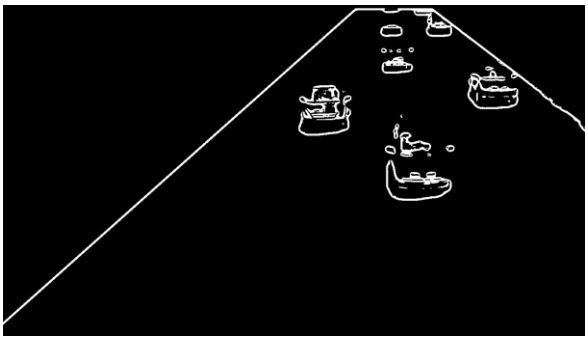






Відфільтроване відео із накладеною маскою:





Програмний код, що забезпечує отримання результату:

Розпізнавання машин на зображенні:

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


def form_input_histogram(img, title):
    if len(img.shape) == 3:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        gray = img

    plt.figure(figsize=(12, 4))
    plt.subplot(121)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(title)

    plt.subplot(122)
    plt.hist(gray.ravel(), 256, [0, 256])
    plt.title('Brightness Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.show()

    return gray


def form_closer_masked_hist(masked_img):
    if len(masked_img.shape) == 3:
        gray = cv2.cvtColor(masked_img, cv2.COLOR_BGR2GRAY)
    else:
        gray = masked_img

        # Plotting the image
    plt.figure(figsize=(15, 5))

    plt.subplot(121)
    plt.imshow(cv2.cvtColor(masked_img, cv2.COLOR_BGR2RGB))
    plt.title('Masked Image')

    hist = cv2.calcHist([gray], [0], None, [256], [0, 256])

    # Ignoring the spike at 0 intensity to see everything else better
    hist[0] = np.nan

    # Plotting the histogram
    plt.subplot(122)
    plt.plot(hist, color='blue')
    plt.title('Brightness Diagram Without Spike at 0')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.xlim([0, 255])
    plt.ylim([0, np.nanmax(hist) * 1.1])
    plt.show()

    return gray
```

```python
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


def histogram_based_filtering(img):
    # Converting to HSV color space
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Defining HSV ranges for vehicle detection
    lower_color1 = np.array([0, 65, 50])  # Light colors
    upper_color1 = np.array([180, 255, 255])

    lower_color2 = np.array([0, 0, 0])  # Dark colors
    upper_color2 = np.array([180, 255, 50])  # Lower V value for darker colors

    # Creating masks using defined color ranges
    mask1 = cv2.inRange(hsv, lower_color1, upper_color1)
    mask2 = cv2.inRange(hsv, lower_color2, upper_color2)

    # Combining masks
    color_mask = cv2.bitwise_or(mask1, mask2)

    # Combining with histogram filtering
    binary = cv2.adaptiveThreshold(
        color_mask, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY_INV, 25, 25
    )

    # Morphological operations to clean up the mask
    kernel = np.ones((2, 2), np.uint8)
    filtered = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
    filtered = cv2.morphologyEx(filtered, cv2.MORPH_OPEN, kernel)

    return filtered


def detect_vehicles(original_img, processed_img, road_mask):
    contours, _ = cv2.findContours(
        processed_img,
        cv2.RETR_EXTERNAL,
```

```
        cv2.CHAIN_APPROX_SIMPLE
    )

    vehicles = []
    for contour in contours:
        area = cv2.contourArea(contour)
        if area < 200:
            continue

        x, y, w, h = cv2.boundingRect(contour)

        # Checking if the contour is within the road mask
        if np.all(road_mask[y:y + h, x:x + w] > 0):
            aspect_ratio = float(w) / h
            if 0.3 < aspect_ratio < 3.0:
                vehicles.append(contour)

    # Drawing detected vehicles
    result = original_img.copy()
    cv2.drawContours(result, vehicles, -1, (0, 255, 0), 2)

    # Drawing rectangles to better see detected vehicles
    for contour in vehicles:
        x, y, w, h = cv2.boundingRect(contour)
        cv2.rectangle(result, (x, y), (x + w, y + h), (255, 0, 0), 2)

    return result, vehicles


def process_frame(frame_path):
    img = cv2.imread(frame_path)
    if img is None:
        raise ValueError(f"Error reading image at {frame_path}")

    form_input_histogram(img, 'Original Image')

    masked_img, road_mask = create_road_mask(img)
    form_input_histogram(masked_img, "Masked image")
    form_closer_masked_hist(masked_img)

    filtered = histogram_based_filtering(masked_img)

    result, vehicles = detect_vehicles(img, filtered, road_mask)

    plt.figure(figsize=(15, 5))
    plt.subplot(131)
    plt.imshow(cv2.cvtColor(masked_img, cv2.COLOR_BGR2RGB))
    plt.title('Masked Road Region')

    plt.subplot(132)
    plt.imshow(filtered, cmap='gray')
    plt.title('Filtered Image')

    plt.subplot(133)
    plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
    plt.title(f'Detected Vehicles: {len(vehicles)}')
    plt.show()

    return result, vehicles
```

```
def main():
    frame_path = 'highway_image.png'
    result, vehicles = process_frame(frame_path)
    print(f"Detected {len(vehicles)} vehicles")


if __name__ == "__main__":
    main()
```

Розпізнавання машин на відео:

```
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 * 0.55)],             # 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


def histogram_based_filtering(img):
    # Converting to HSV color space
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Defining HSV ranges for vehicle detection
    lower_color1 = np.array([0, 60, 50])  # Light colors
    upper_color1 = np.array([180, 255, 255])

    lower_color2 = np.array([0, 0, 0])  # Dark colors
    upper_color2 = np.array([180, 255, 50])  # Lower V value for darker colors

    # Creating masks using defined color ranges
    mask1 = cv2.inRange(hsv, lower_color1, upper_color1)
    mask2 = cv2.inRange(hsv, lower_color2, upper_color2)

    # Combining masks
    color_mask = cv2.bitwise_or(mask1, mask2)

    # Combining with histogram filtering
    binary = cv2.adaptiveThreshold(
        color_mask, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
```

```
        cv2.THRESH_BINARY_INV, 25, 25
    )

    # Morphological operations to clean up the mask
    kernel = np.ones((2, 2), np.uint8)
    filtered = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
    filtered = cv2.morphologyEx(filtered, cv2.MORPH_OPEN, kernel)

    return filtered


def detect_vehicles(original_img, processed_img, road_mask):
    contours, _ = cv2.findContours(
        processed_img,
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE
    )

    vehicles = []
    for contour in contours:
        area = cv2.contourArea(contour)
        if area < 200:
            continue

        x, y, w, h = cv2.boundingRect(contour)

        # Checking if the contour is within the road mask
        if np.all(road_mask[y:y + h, x:x + w] > 0):
            aspect_ratio = float(w) / h
            if 0.3 < aspect_ratio < 3.0:
                vehicles.append(contour)

    # Drawing detected vehicles
    result = original_img.copy()
    cv2.drawContours(result, vehicles, -1, (0, 255, 0), 2)

    # Drawing rectangles to better see detected vehicles
    for contour in vehicles:
        x, y, w, h = cv2.boundingRect(contour)
        cv2.rectangle(result, (x, y), (x + w, y + h), (255, 0, 0), 2)

    return result


def process_video(video_path):
    cap = cv2.VideoCapture(video_path)

    if not cap.isOpened():
        print("Error: Could not open video.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break  # Exit the loop if there are no more frames

        masked_img, road_mask = create_road_mask(frame)
        filtered = histogram_based_filtering(masked_img)
        result = detect_vehicles(frame, filtered, road_mask)
```

```
        # Display the original frame with detected vehicles
        cv2.imshow('Detected Vehicles', result)

        # Display the filtered image
        cv2.imshow('Filtered Video', filtered)

        # Exit the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the video capture and close windows
    cap.release()
    cv2.destroyAllWindows()


def main():
    video_path = 'highway_video.mp4'
    process_video(video_path)


if __name__ == "__main__":
    main()
```

Висновки: виконавши цю лабораторну роботу, я навчилася ще ефективніше застосовувати фільтри до зображень та покращувати їхню якість, ідентифікуючи контури об’єктів достатньо чітко. Також я попрактикувалась накладати маски на зображення, зрозуміла як «читати» гістограми яскравості і як працювати із відеопотоками в рамках задач Computer Vision. Лабораторна була дуже інформативною, хоча і дещо складною через необхідність підлаштування деяких параметрів та порогів.