Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Звіт

Лабораторна робота № 3 з дисципліни
«Штучний інтелект в задачах обробки зображень»

«Розмітка дорожньої лінії засобами OpenCV»

Виконав(ла) *ІП-01 Черпак А. В.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив(ла) *Нікітін В. А.* _____
(прізвище, ім'я, по батькові)

Київ 2023

Завдання

1. Проробити з будь-якою фотографією процедури, які описані в теоретичних відомостях;
2. Зробити розпізнавання розмітки з будь-якого відеофайлу.

Хід роботи

1. Імпортуємо необхідні бібліотеки та створимо допоміжну функцію для виведення зображення:

```
import numpy as np
import cv2


def show(label: str, image_to_show: np.ndarray) -> None:
    cv2.imshow(label, image_to_show)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2. Створимо функція для демонстрації розглянутих у теоретичних відомостях кроків та їх результатів:

```
def demonstrate_staps():
    img = cv2.imread("road2.jpg")
    show("original image", img)

    # конвертуємо зображення у чорнобіле
    grayScale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    show("grayscale image", grayScale)

    # розмиваємо зображення для того, аби прибрати шуми
    kernel_size = 5
    blur = cv2.GaussianBlur(grayScale, (kernel_size, kernel_size), 0)
    show("blur image", blur)

    # за допомогою алгоритму Кенні знаходимо межі об'єктів
    low_t = 50
    high_t = 150
    edges = cv2.Canny(blur, low_t, high_t)
    show("edges image", edges)

    # Оскільки нас цікавлять лише об'єкти у межах певної області, накладемо на
зображення маску
    vertices = np.array(
        [[(0, img.shape[0]), (450, 310), (490, 310), (img.shape[1],
img.shape[0])]], dtype=np.int32)
    mask = np.zeros_like(edges)
    cv2.fillPoly(mask, vertices, 255)
    show("mask image", mask)
    masked_edges = cv2.bitwise_and(edges, mask)
    show("masked_edges image", masked_edges)
```

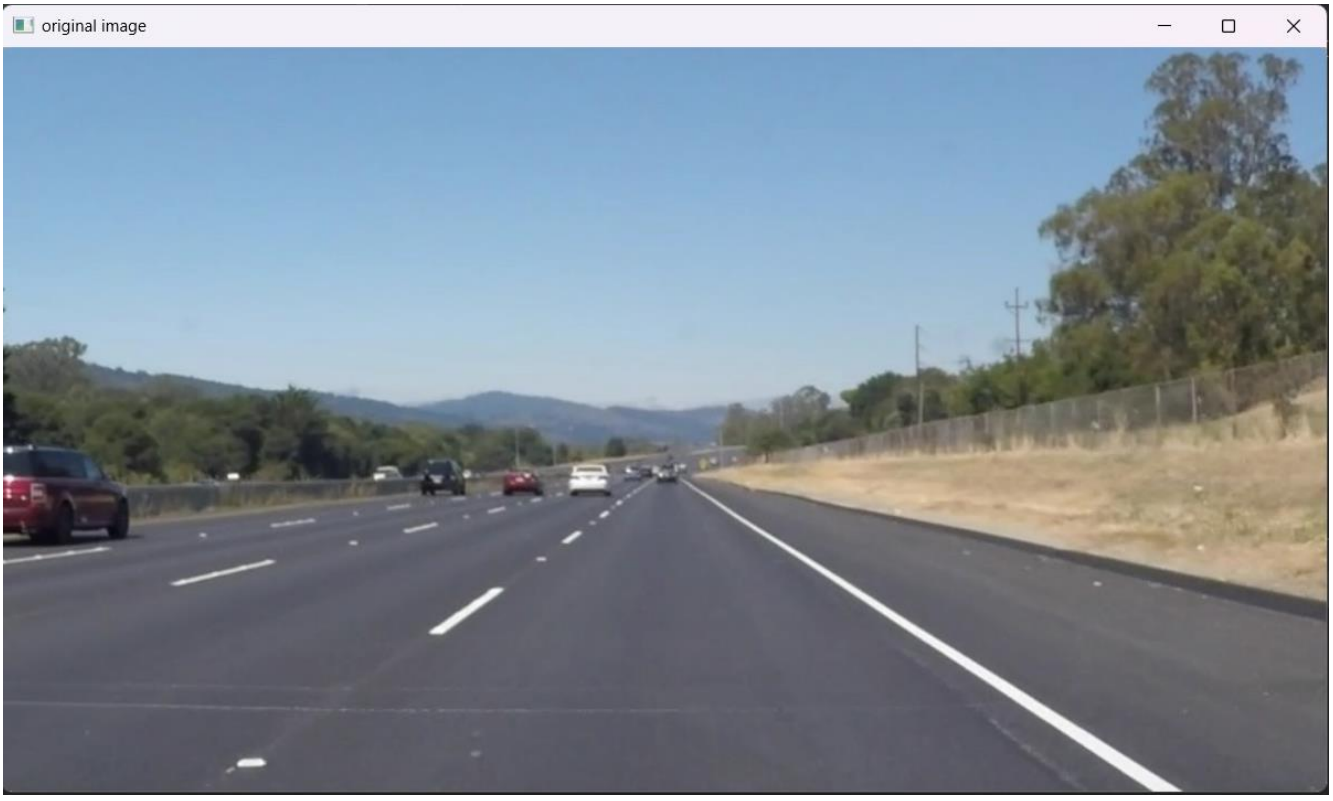

Рисунок 1 – Оригінальне зображення

Рисунок 2 – Конвертоване в сіре зображення

Рисунок 3 – Розмите зображення

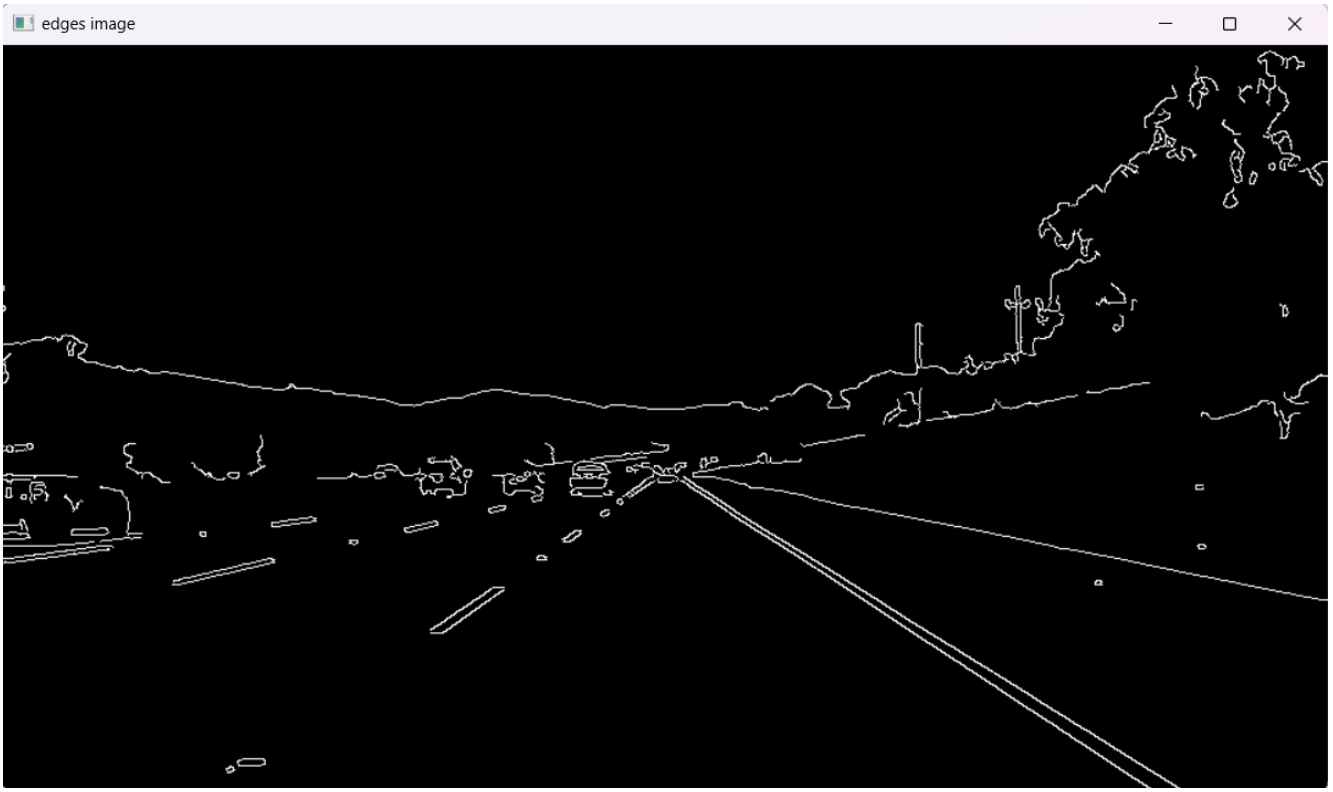

Рисунок 4 – Межі об’єктів, знайдені за допомогою алгоритму Кенні

Рисунок 5 – Маска, накладена на зображення

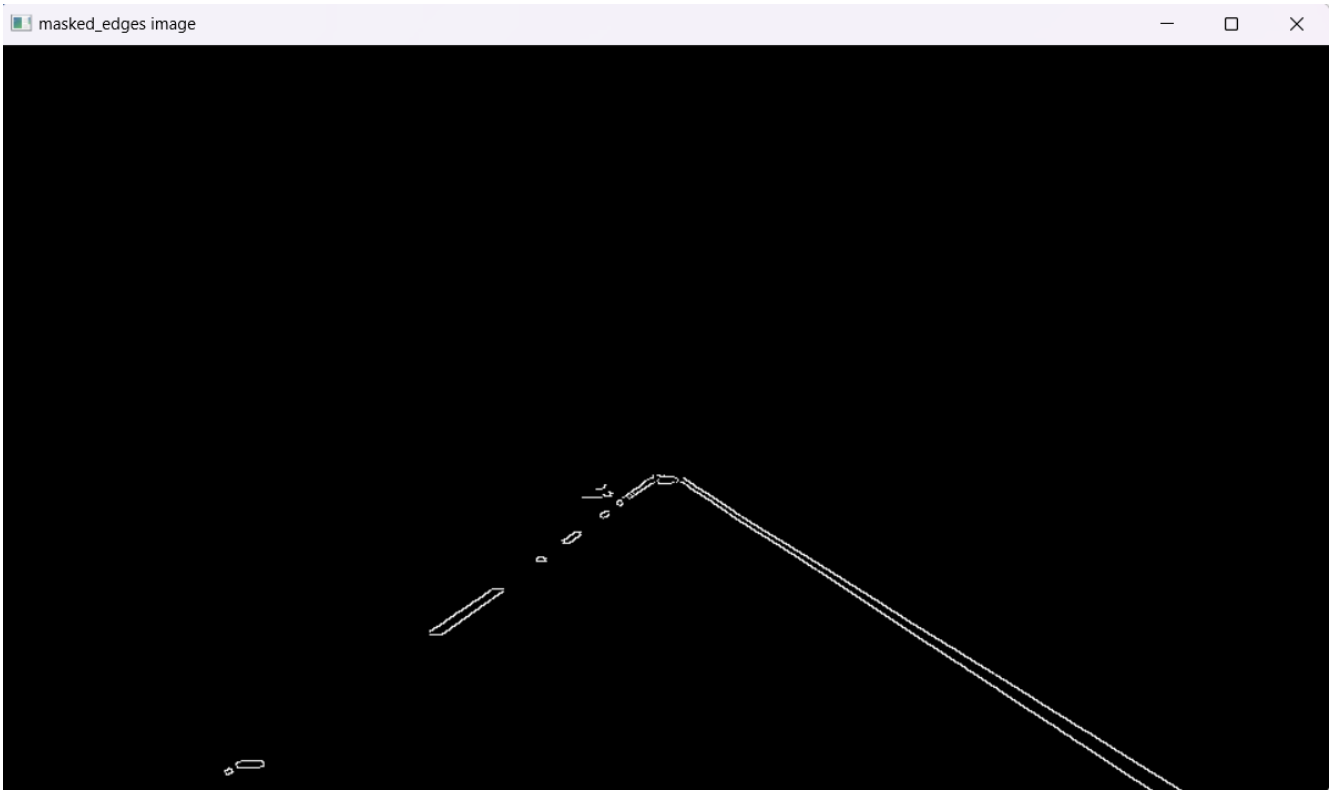

Рисунок 6 – Виділені маскою межі об’єктів

3. Перетворення Хафа та обробка відеофайлу:

```
4. # Метод для малювання ліній дорожньої розмітки на зображенні
   def draw_lines(frame, lines, color=[0, 0, 255], thickness=10):
       x_bottom_pos = []
       x_upper_pos = []
       x_bottom_neg = []
       x_upper_neg = []
       y_bottom = 540
       y_upper = 315
       for line in lines:
```

```
            for x1, y1, x2, y2 in line:
                slope = ((y2 - y1) / (x2 - x1))
                b = y1 - slope * x1
                if slope > 0.5 and slope < 0.8:
                    x_bottom_pos.append((y_bottom - b) / slope)
                    x_upper_pos.append((y_upper - b) / slope)
                elif slope < -0.5 and slope > -0.8:
                    x_bottom_neg.append((y_bottom - b) / slope)
                    x_upper_neg.append((y_upper - b) / slope)
        if len(x_bottom_pos) > 0 and len(x_bottom_neg) > 0:
            lines_mean = np.array(
                [[int(np.mean(x_bottom_pos)), int(np.mean(y_bottom)),
    int(np.mean(x_upper_pos)), int(np.mean(y_upper))],
                 [int(np.mean(x_bottom_neg)), int(np.mean(y_bottom)),
    int(np.mean(x_upper_neg)), int(np.mean(y_upper))]])
            for i in range(len(lines_mean)):
                cv2.line(frame, (lines_mean[i, 0], lines_mean[i, 1]),
                         (lines_mean[i, 2], lines_mean[i, 3]), color, thickness)


    # Метод для обробки зображення - пошуку на ньому дорожньої розмітки
    def process_image(frame):
        vertices = np.array(
            [[(0, frame.shape[0]),
              (450, 310),
              (490, 310),
              (frame.shape[1], frame.shape[0])]], dtype=np.int32)
        grayScale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(grayScale, (5, 5), 0)
        edges = cv2.Canny(blur, 50, 150)
        mask = np.zeros_like(edges)
        cv2.fillPoly(mask, vertices, 255)
        masked_edges = cv2.bitwise_and(edges, mask)
        lines = cv2.HoughLinesP(
            masked_edges, 3, np.pi / 180, 15, np.array([]),
            minLineLength=100,
            maxLineGap=70)
        draw_lines(frame, lines)


    # метод для покадрової обробки та відображення відео
    def process_video():
        cv2.startWindowThread()
        video_capture = cv2.VideoCapture("road.mp4")
        while video_capture.isOpened():
            ret, frame = video_capture.read()
            if ret:
                process image(frame)
                cv2.imshow("frame", frame)
                if cv2.waitKey(1) & 0xFF == ord("q"):
                    break
            else:
                break
        video_capture.release()
        cv2.destroyAllWindows()
```

Рисунок 7 – Кадр з обробленного відеофайлу

Контрольні запитання:

1. Частіше за все межі об’єктів можна визначити зі змін яскравості. Саме тому інколи вигідніше конвертувати зображення у чорно-біле і працювати лише з одним каналом. Це значно зекономить нам пам’ять та обчислювальні ресурси. Оскільки у даній задачі нам необхідно відокремити контури білих об’єктів на сірому фоні, інформація про відтінок кольору зовсім не має для нас значення, тому чорно-біле зображення ідеально підійде.
2. При розмиванні зображення прибираються дрібні деталі, у тому числі й шуми. Це дозволяє зменшити частоту появи хибних контурів, а отже і збільшити точність розпізнавання. Насправді у нашому випадку шуми практично відсутні, тому розмивання не дасть особливої користі. Проте й ніяк не завадить, оскільки лінії розмітки будуть чудово помітні у будь-якому випадку.
3. Алгоритм Кенні визначає межі об’єктів шляхом обчислення градієнтів інтенсивності зображення. Потім за допомогою двох порогових значень відсікаються незначні межі.
4. Перетворення Хафа - це алгоритм пошуку об’єктів, що належать певному класу фігур. Він працює шляхом пошуку простих фігур, таких як лінії, кола і т.д.
5. Маска допомагає обмежити область пошуку об’єкта. Часто ми наперед знаємо, що частина отриманого зображення не може містити необхідний нам об’єкт. Тоді є сенс обмежити область пошуку, аби не опрацьовувати завідома непотрібні ділянки зображення. Окрім економії обчислювальних ресурсів, це дозволить також зменшити кількість фальшивих розпізнавань. У нашій ситуації необхідно шукати виключно лінії розмітки перед машиною та ігнорувати будь-які інші лінії.