«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»

Кафедра «Высшая математика»

КУРСОВАЯ РАБОТА

на тему: «Исследование характеристик транспортного потокаа по видеоряду с камеры наблюдения»

по дисциплине: «Прикладное программирование и пакеты программ»

Выполнил

Учебная группа: 1бПМ2

Тихонов Владислав Анатольевич

Подпись _____

Руководитель:

Старший преподаватель

Доткулова Анастасия Сергеевна

Подпись _____

Москва 2022 г.

Содержание

1 Введение

1.1 Описание работы

Цель курсовой работы - написать программу, способную исследовать характеристики транспортного потока по видеоряду с помощью компьютерного зрения.

Такими харектиристиками являются:

- Цвет исследуемых участков видеоряда;
- Количество автомобилей;
- Интенсивность потока.

Исходными данными является запись вэбкамеры по адресу Ленинградский пр-т., 64, Москва. Запись велась днём при облачной погоде. Актуальность работы заключается в распространении участия компьютерного зрения в различных направлениях IT-продуктов.

1.2 Ход работы

Ход работы можно разбить на пукты. Сначала требуется выбрать инструменты для работы. Самые изветсные и доступные tools для работы с компьютерным зрением - OpenCV и MATLAB. MATLAB удобнее в разработке и предоставлении данных, однако OpenCV намного быстрее в исполнении. В данном случае используется OpenCV на Python.

Далее получаем видеопоток и выполняем требуемый анализ. Программа получает покадровое изображение и работает с каждым фреймом отдельно, из-за этого при некоторых задачах время работы программы довольно продолжительное.

Для распознавания автомобилей используется вспомогательный .xml файл, который можно найти в документации к OpenCV.

2 Программа

2.1 Код программы

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


def show():
    cap = cv2.VideoCapture("video.mp4")
    while True:
        ret, frame = cap.read()
        if ret:
            cv2.imshow('WebCam', frame)
        else:
            break
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()


def make_gray_frames():
    cap = cv2.VideoCapture("video.mp4")
    frame = cap.read()[1]
    cv2.imshow('frame', frame)

    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow('gray_frame', gray_frame)

    threshold_frame = cv2.threshold(gray_frame, 127, 255, 0)[1]
    cv2.imshow('threshold_image', threshold|verb||frame)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def detector():
```

```
    cap = cv2.VideoCapture("video.mp4")
    while True:
        frame = cap.read()[1]
        car_cascade = cv2.CascadeClassifier('cars.xml')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cars = car_cascade.detectMultiScale(gray, 1.1, 1)
        for (x, y, w, h) in cars:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
        plt.figure(figsize=(10, 20))
        plt.clf()
        cv2.imshow("frame frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()


def colors_graphs():
    cap = cv2.VideoCapture("video.mp4")
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        color = ('b', 'g', 'r')
        car_cascade = cv2.CascadeClassifier('cars.xml')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cars = car_cascade.detectMultiScale(gray, 1.1, 1)
        for (x, y, w, h) in cars:
            mask = np.zeros(frame.shape[:2], np.uint8)
            mask[y:y + h, x:x + h] = 255
            for i, col in enumerate(color):
                masked_img = cv2.bitwise_and(frame, frame, mask=mask)
                hist_mask = cv2.calcHist([frame], [i], mask, [256], [0, 256])
                plt.subplot(221), plt.imshow(frame, 'gray')
                plt.subplot(222), plt.imshow(mask, 'gray')
                plt.subplot(223), plt.imshow(masked_img, 'gray')
```

```
            plt.subplot(224), plt.plot(hist_mask, color=col)
            plt.xlim([0, 256])
        plt.show()
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


def binary_graphs():
    cap = cv2.VideoCapture("video.mp4")
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        threshol_frame = cv2.threshold(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), 127,
255, 0)[1]
        plt.hist(gray_frame.ravel(), 256, [0, 256])
        plt.show()
        plt.close()
        plt.hist(threshold_frame.ravel(), 256, [0, 256])
        plt.show()
        cv2.waitKey(0)
    cv2.destroyAllWindows()


def colors_and_intensity():
    cap = cv2.VideoCapture("video.mp4")
    color = [[], [], []]
    total = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        car_cascade = cv2.CascadeClassifier('cars.xml')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cars = car_cascade.detectMultiScale(gray, 1.1, 1)
```

```
        for (x, y, w, h) in cars:
            mask = np.zeros(frame.shape[:2], np.uint8)
            mask[y:y + h, x:x + h] = 255
            r, g, b, w = cv2.mean(frame, mask)
            color[0].append(r)
            color[1].append(g)
            color[2].append(b)
            print(r, g, b)
            total += 1
    print("Средний цвет: sum(color[0]) / len(color[0]), sum(color[1]) / len(color[1]), sum(color[2])
/ len(color[2]))
    print("Примерное количество машин: (total / 90) * 2)
    print("Примерная интенсивность потока (машин/мин): (total / 90 / 37) * 60 * 2)
    x = [i for i in range(total)]
    y1 = color[0]
    y1 = color[1]
    y1 = color[2]
    fig, ax = plt.subplots()
    ax.plot(x, y1, color="red")
    ax.plot(x, y2, color="green")
    ax.plot(x, y3, color="blue")
    ax.set_xlabel("timeline")
    ax.set_ylabel("RGB")
    plt.show()


show()
make_gray_frames()
detector()
colors_graphs()
binary_graphs()
colors_and_intensity()
```

2.2 Комментарии по коду

В коде используются следующие библиотеки: cv2 - openCV, numpy - для создания масок, matplotlib - для рисования графиков; следующие функции: show() - для вывода видеопотока, make_gray_frames() - для конвертации кадров в градации серого, detector() - для распознавания автомобилей, colors_graphs() - для создания RGB-графика для каждого детектора, binary_graphs() - для создания бинаризированного графика для каждого детектора, colors_and_intensity() - для нахождения среднего цвета и анализа транспортного потока.
Некоторые важные замечания для понимания кода:

- В функции detector() метод cvtColor конвертирует кадр в градацию серого, это нужно для лучшего распознавания объектов;
- В той же функции метод detectMultiScale обнаруживает автомобиль и возвращает границы отсека, в котором он находится;
- В функции colors_graphs() метод calcHist высчитывает гистограмму кадра;
- В функции colors_and_intensity() при вычислении интенсивности потока количество вхождений границ автомобилей делится на 90 и умножается на 2. В этом случае считается, что один и тот же автомобиль программа распознаёт с среднем 90 раз и при этом видит только одну полосу движения.

3 Заключение

В результате написания программы получаем продукт, подходящий для анализа транспотного потока. Важно отметить, что в коде используется динамичное распознавание объектов, которое намного ближе к професиональным CV продуктам, чем заранее определённая область анализа.

Список литературы

[1] Документация библиотеки OpenCV - https://docs.opencv.org