Федеральное государственное автономное образовательное учреждение
высшего образования «Университет ИТМО»

Факультет программной инженерии и компьютерной техники

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине «Компьютерное зрение»

Выполнил: Федотовских Е.А.

Группа: 4141

Санкт-Петербург

2022

Задание: 1. Реализовать программу согласно варианту задания. Базовый алгоритм, используемый в программе, необходимо реализовать в 3 вариантах: с использованием встроенных функций какой-либо библиотеки (OpenCV, PIL и др.) и нативно на Python + |с использованием Numba или C++|. 2. Сравнить быстродействие реализованных вариантов.

Примечание. Программа работает с видео. На вход должен поступать видеопоток с устройства (камеры) или видео должно читаться из файла. Каждый алгоритм нужно реализовать в 3 вариантах: с использованием сторонних библиотек на Python, с помощью примитивных операций и циклов на Python (можно использовать NumPy массивы) и с помощью компилируемого кода (на C++ либо с использованием, например, Numba на Python). Если указано, что выходное изображение переключается между черно-белым и после обработки, это значит, что на вход обработки подается черно-белое изображение.

Вариант задания:

Бинаризация с адаптивным порогом. На вход поступает изображение, программа отрисовывает окно, в которое выводится либо исходное изображение после преобразования в черно-белое, либо после бинаризации (переключение по нажатию клавиши).

Теория:

Суть бинаризации заключается в определении порога яркости оттенков серого для картинки. Если значение яркости пикселя выше порогового, то пиксель заменяется на черный (255), ниже – на белый (0).

Существует бинаризация с простым порогом и адаптивным. В данной лабораторной необходимо реализовать бинаризацию с адаптивным порогом методом среднего. Для этого все изображение разбивается на блоки размером blocksize* blocksize, для которого считается среднее значение яркости пикселя, затем из него вычитается константа, единая для каждого из блоков. Также можно домножать получившееся среднее для каждого блока на общую вторую константу const2.

Реализация:

Программа считывает видео из файла с помощью функции OpenCV. Затем идет цикл по кадрам, в котором для каждого кадра происходит переход в оттенки серого, а затем происходит бинаризация изображения. Далее этот кадр выводится на экран, а также сохраняется с помощью функции VideoWriter, которая принимает на вход кадры и сохраняет их в видео.

Реализация с помощью OpenCV:

```
start = time.time()
cap = cv.VideoCapture('Narrow_Neck_BW_1.mov')
fl_to_show=1
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('output.avi', fourcc, 20.0, (1440,  810),False)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    frame = cv.resize(frame, (1440, 810))
    frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
#     ret_prep,thresh = cv.threshold(frame,127,255,cv.THRESH_BINARY)
#     ret_prep,thresh = cv.threshold(frame,127,255,cv.THRESH_BINARY_INV)
    thresh = cv.adaptiveThreshold(frame,255,cv.ADAPTIVE_THRESH_MEAN_C,\
                                  cv.THRESH_BINARY,11,2)
    out.write(thresh)
#     thresh = cv.adaptiveThreshold(frame,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,\
#             cv.THRESH_BINARY,11,2)
    if cv.waitKey(1) == ord('2'):
        if fl_to_show==1: fl_to_show = 2
        else: fl_to_show = 1
    if fl_to_show==1:
        frame_to_show = frame
    else:
        frame_to_show = thresh

    cv.imshow('frame', frame_to_show)
    if cv.waitKey(1) == ord('q'):
        break
cap.release()
out.release()
cv.destroyAllWindows()
end = time.time()
print(round((end-start),2))
```

Рисунок 1 – Листинг главного цикла

Функцию бинаризации изображения без готовых решений можно реализовать с помощью циклов.

```
def adaptive_binarization_mean(frame,blockSize, constant, const2):
    height = np.shape(frame)[0]//blockSize
    width = np.shape(frame)[1]//blockSize
    frame_binary = frame
    for h in range(height):
        for w in range(width):
            s = 0
            for i in range(blockSize):
                for j in range(blockSize):
                    s = s + frame[h*blockSize+i][w*blockSize+j]
            threshhold = const2*(s/(blockSize**2)) - constant
            for i in range(blockSize):
                for j in range(blockSize):
                    if frame_binary[h*blockSize+i][w*blockSize+j]>threshhold:
                        frame_binary[h*blockSize+i][w*blockSize+j]=255
                    else: frame_binary[h*blockSize+i][w*blockSize+j] = 0

    return frame_binary
```

Рисунок 2 – Листинг функции бинаризации

Результаты работы программы:

Рисунок 3 – Исходное изображение

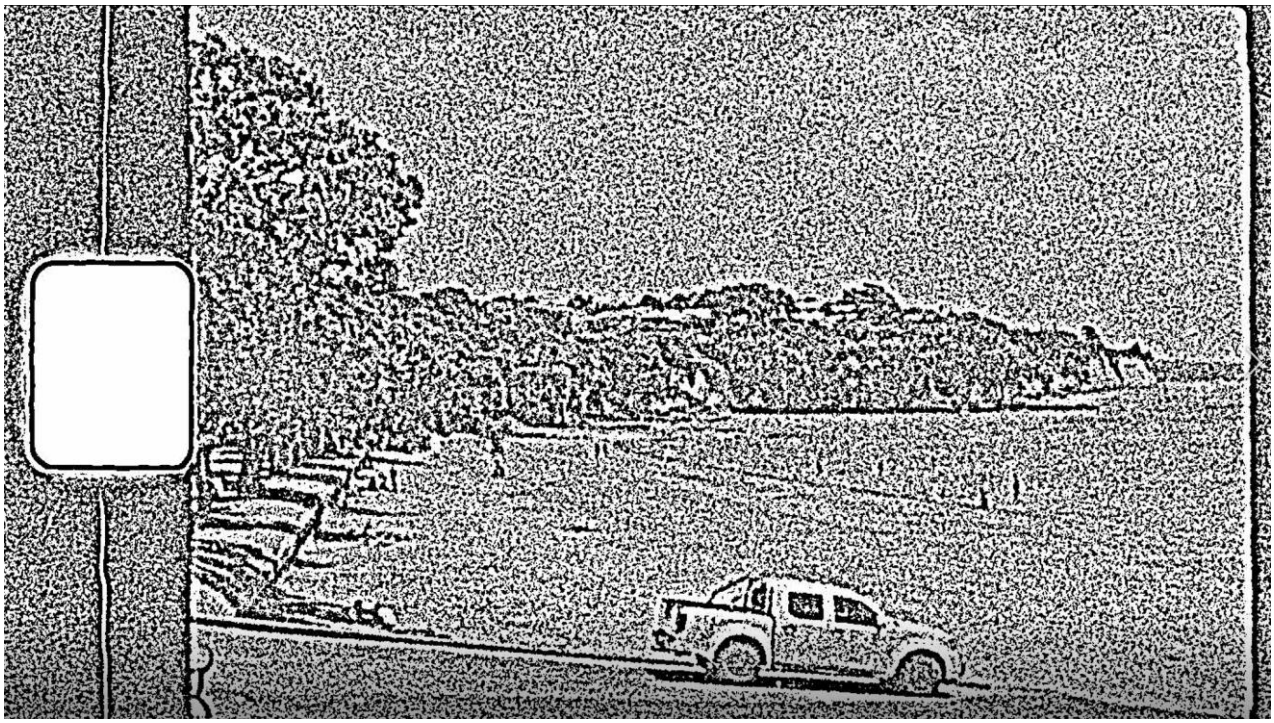

Рисунок 4 –Изображение обработанное с OpenCV

Рисунок 5 –Изображение обработанное с помощью своей функции

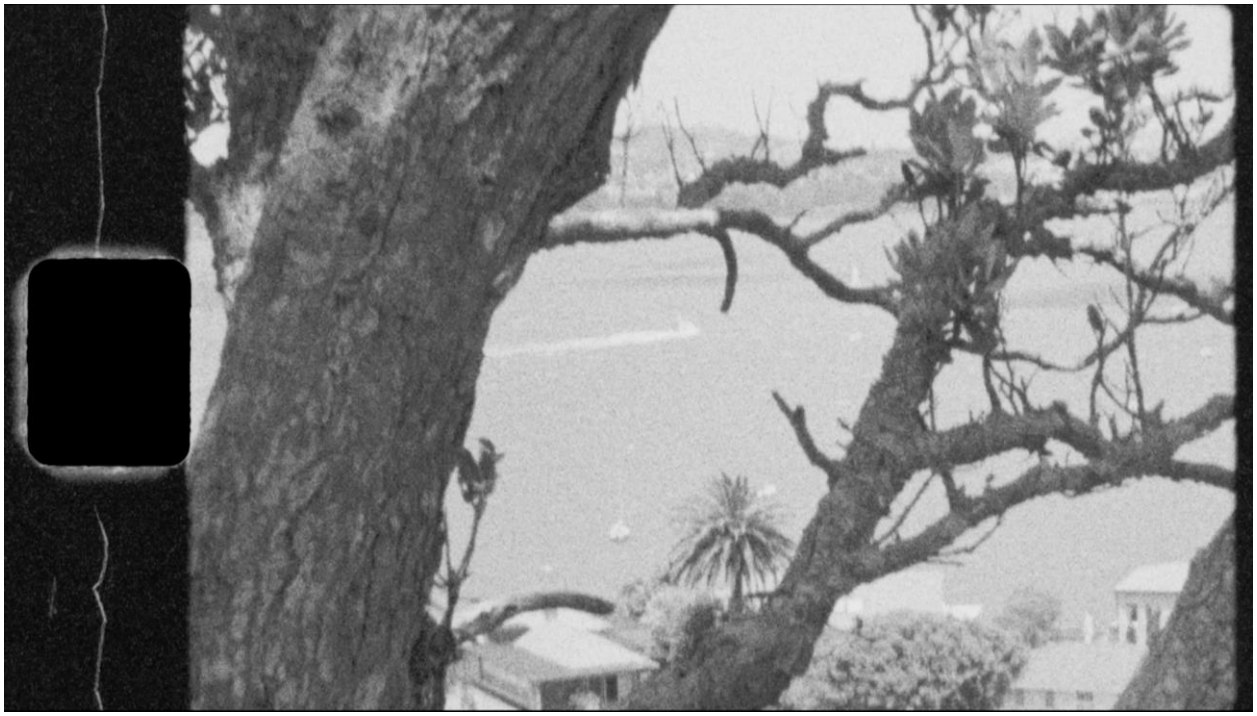

Рисунок 6 – Исходное изображение 2

Рисунок 7 –Изображение 2 обработанное с OpenCV

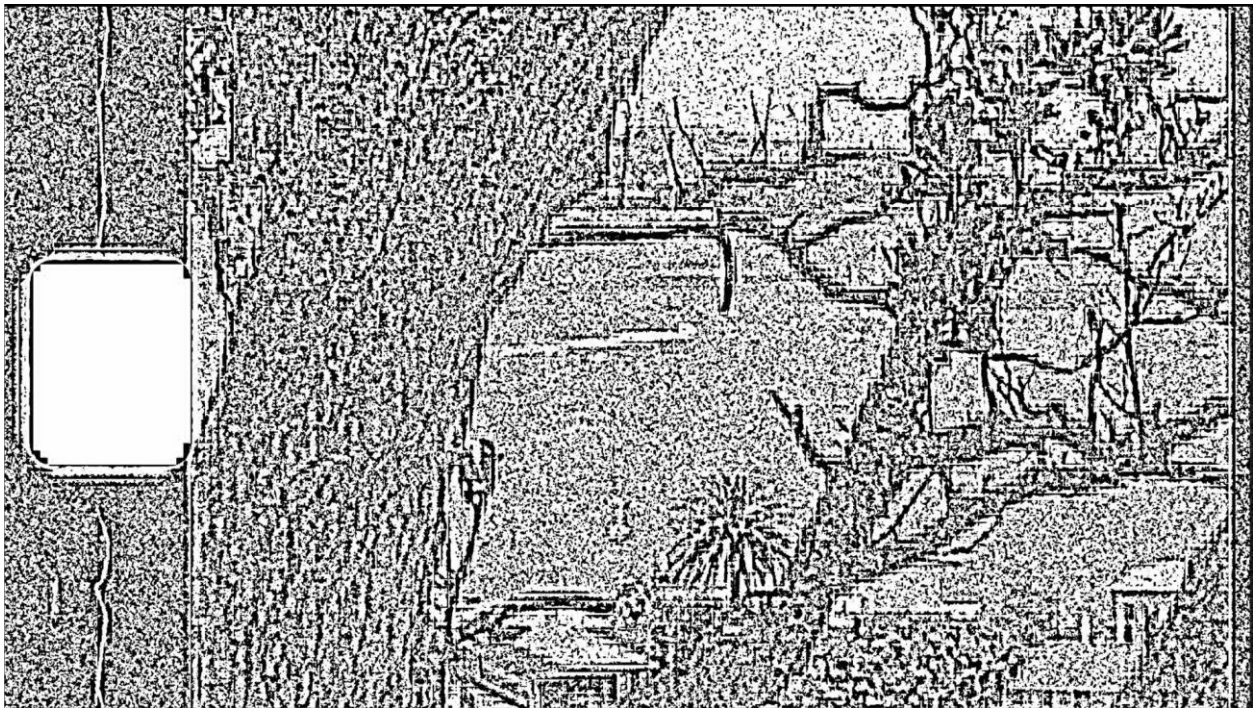

Рисунок 8 –Изображение 2 обработанное с помощью своей функции

Теперь попробуем вычесть обработанные изображения разными методами друг из друга.

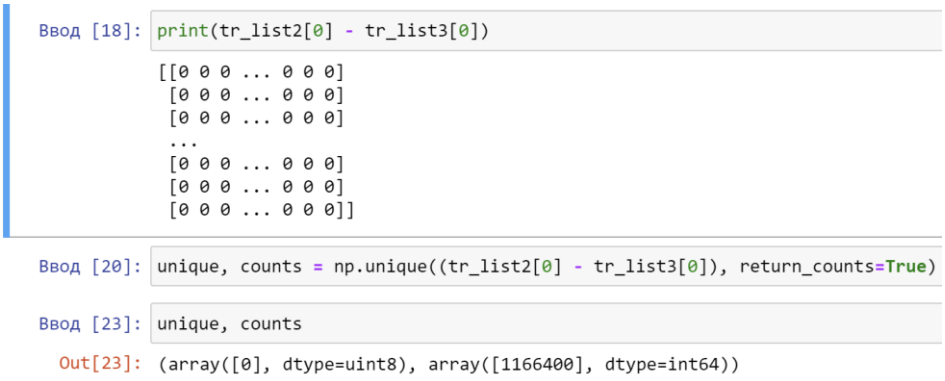

Рисунок 9 –Разница между методами с циклами и с использованием numba

Как можно видеть, изображения при обработке с циклами и с циклами, но скомпилированном кодом совпадают полностью.

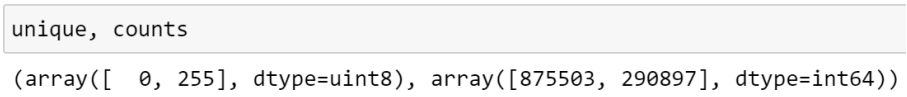

Рисунок 10 –Разница между методами с циклами с функцией OpenCV

Видно, что не все пиксели совпали, а только 75%.

Выводы:

В результате работы была реализована бинаризация видео 3 вариантами: с использованием сторонних библиотек на Python, с помощью примитивных операций и циклов на Python и с помощью компилируемого кода с использованием Numba.

Время выполнения бинаризации одного и того же видео:

С использованием OpenCV: 7.7 сек

Нативно на python: 1440 сек

Компилируемый код: 9.65 сек

Скомпилированный код работает немного медленнее, чем функция OpenCV, это связанно с тем, что в функции грамотно распараллелены вычисления.

По картинкам видно, что они немного отличаются друг от друга.

При этом обработанные картинки обоими вариантами выглядят «адекватно», но при использовании готового решения не так заметны квадратные области, для каждой из которых рассчитывается свой порог. Это, скорее всего, связанно с тем, что в функции OpenCV происходит какая-то последующая обработка изображения.