МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА АЭРОКОСМИЧЕСКИХ КОМПЬЮТЕРНЫХ И ПРОГРАММНЫХ СИСТЕМ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Старший преподаватель		А. В. Борисовская
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1
ИЗУЧЕНИЕ ФОРМАТА AVI

по курсу: МУЛЬТИМЕДИА ТЕХНОЛОГИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5722		Е. Д. Энс
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1. Цели работы

- Ознакомление со структурой файла в формате AVI
- Анализ статистических свойств видеопоследовательностей
- Получение практических навыков работы с видеопоследовательностями

2. Описание структуры формата AVI

Формат AVI используется для хранения видео, как совокупности аудиоданных и последовательности изображений. AVI файлы могут содержать несколько потоков с различными данными, но большинство содержат два – одно аудио и одно видео.

В качестве контейнерного AVI используют формат RIFF. Элемент формата RIFF начинается с четырёхбуквенного кода, идентификатора элемента. За идентификатором следует двойное слово, определяющее длину данных, которые содержатся в элементе. За ними идут байты данных, относящихся к элементу. Всего имеется два основных типа элементов:

1. Порция (chunk).
   Синтаксис:
   ckID ckSize ckData
   где ckID – FOURCC-идентификатор
   ckSize – размер порции в байтах
   ckData – данные, относящиеся к порции, если они есть.

2. Список (list)
   Синтаксис:
   ‘LIST’ listSize listType listData
   где ‘List’ – четырёхбуквенный код, идентифицирующий элемент,как список
   listSize – размер списка в байтах
   listType – FOURCC код, интерпретируемый как имя списка
   listData – данные, представляющие собой последовательную запись элементов RIFF.

Помимо перечисленных типов, также следует выделить заголовок файла в формате RIFF. Это порция, но её синтаксис аналогичен списку, за исключением того, что вместо четырёхбуквенного кода ‘LIST’ используется ‘RIFF’:

‘RIFF’ fileSize fileType aviData
где fileType является четырехбуквенным кодом ‘AVI’
а под aviData понимаются остальные данные файла.

Все AVI файлы включают два обязательных списка, которые определяют формат потока и поток данных соответственно. Таким образом файл имеет следующую форму:

RIFF (‘AVI ’
LIST (‘hdrl’ …)
LIST (‘movi’ …)
[‘idxl’ (<AVI Index>)]
)

Первый обязательный список – ‘hdrl’ описывает формат данных. Второй обязательный список – ‘movi’ содержит данные. Список ‘idxl’ содержит индекс. AVI файл должен содержать эти списки в правильной последовательности. Списки ‘hdrl’ и ‘movi’ используют вложенные порции (chunks) для представления данных.

3. Алгоритм работы с файлом AVI

Для работы с файлом в формате AVI используется обвязка pyAV. Далее осуществляются следующие шаги:

- Открытие/создание всех используемых файлов (входных и выходных)
- Открытие входного видеопотока
- Определение информации о потоке и формате кадров
- Создание выходного видеопотока, информация о потоке получена на предыдущем шаге
- Задание формата кадров для созданного потока, формат был получен ранее
- Чтение кадров из входного потока, их обработка и запись в выходной поток
- Закрытие всех открытых потоков и файлов

4. Краткое описание используемых функций

Для чтения данных был реализован метод **in_video(input)**, который принимает AVI-файл в качестве аргумента. Данный метод возвращает следующие параметры: stream, W, H, format, codec, rate, frames

- где stream – поток данных,
- W – ширина кадра,
- H – высота кадра,
- format – формат хранения данных,
- codec – используемый кодек,
- rate – битрейт потока,
- frames - список с декодированными кадрами из видеопотока

Для записи данных был реализован метод **out_video(name_video, W, H, format, codec, rate)**:

- где name_video – название выходного файла,
- W – ширина кадра,
- H – высота кадра,
- format – формат хранения данных,
- codec – используемый кодек,
- rate – битрейт потока

Данный метод возвращает output, stream_out

- где output – выходной файл
- stream_out – поток для записи в него.

Для записи в поток необходимо закодировать кадры с помощью метода **stream_out.encode()**, а затем записать этот поток в AVI файл методом **output.mux**(). После записи нужно закрыть все потоки записи и файл.

5. Исходные данные

Кадры из фильма LR1_1.avi

Рис.1 – Первый кадр файла LR1_1.avi

Рис.2 – Кадр из середины файла LR1_1.avi

Рис.3 – Последний кадр из файла LR1_1.avi

Кадры из фильма LR1_2.avi

Рис.4 – Первый кадр файла LR1_2.avi

Рис.5 – Кадр из середины файла LR1_2.avi

Рис.6 – Последний кадр из файла LR1_2.avi

Кадры из фильма LR1_3.avi

Рис.7 – Первый кадр из файла LR1_3.avi

Рис.8 – Кадр из середины файла LR1_3.avi

Рис.9 – Последний кадр из файла LR1_3.avi

6. Задания по изменению порядка кадров

Задание 1: сформировать выходной файл, который будет содержать кадры входного фильма в обратном порядке.

Метод, который выполняет данное задание:

```
output, stream_out = out_video('reversed.AVI', W_1, H_1, format_1, codec_1,
rate_1)
tmp_frames = list(frames_1)
for i in range(len(frames_1) // 2):
    tmp = tmp_frames[i]
    tmp_frames[i] = tmp_frames[len(frames_1) - i - 1]
    tmp_frames[len(frames_1) - i - 1] = tmp

for frame in (tmp_frames):
    image = av.VideoFrame.from image(Image.fromarray(frame, mode='RGB'))
    for packet in stream_out.encode(image):
        output.mux(packet)

output.close()
```

Задание 2: сформировать выходной файл, который будет сначала содержать кадры первого входного фильма, а после них будут храниться кадры второго файла.

Метод, который выполняет данное задание:

```
output, stream_out = out_video('concat.AVI', W_1, H_1, format_1, codec_1,
rate_1)
for frame in frames_1 + frames_2:
    image = av.VideoFrame.from_image(Image.fromarray(frame, mode='RGB'))
    for packet in stream_out.encode(image):
        output.mux(packet)

output.close()
```

7. Графики автокорреляции функции для нескольких типов фильмов

Чтобы получить значения коэффициента автокорреляции используется следующая формула:

$$r_{S_t,S_{t+1}} = \frac{\overline{(S_t - \overline{S_t}) \times (S_{t+1} - \overline{S_{t+1}})}}{\sigma_{S_t} \times \sigma_{S_{t+1}}}$$

Полученные значения коэффициента корреляции для разных кадров видео позволяют оценить “схожесть” кадров между собой, то есть оценить временную избыточность кадров.

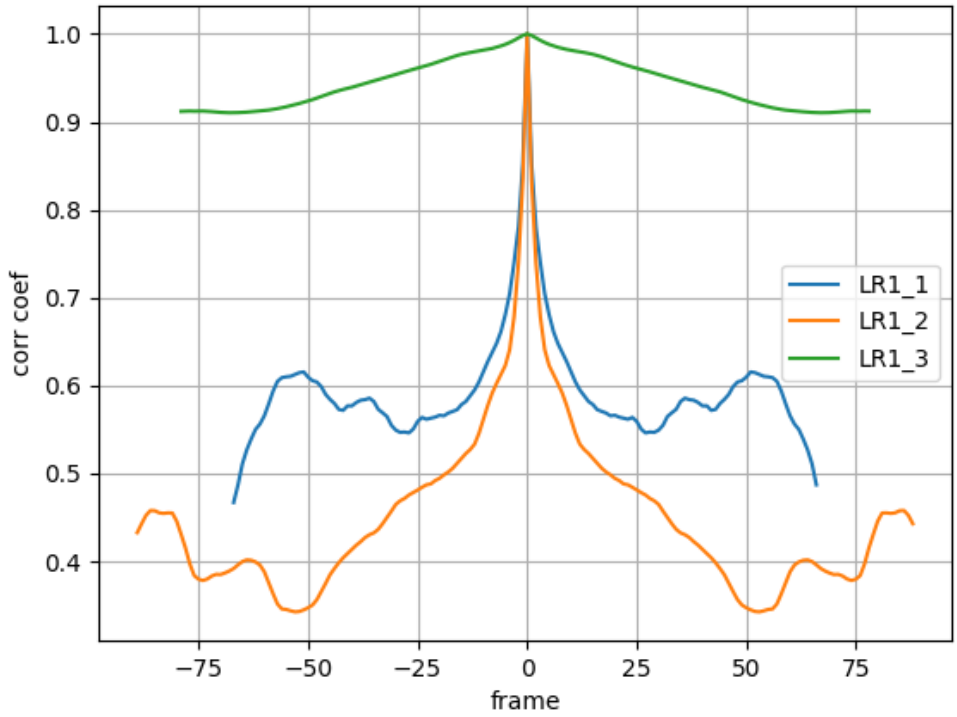

Рис. 10 – график автокорреляци

По полученному графику можно сказать, что из всех представленных наименьшая корреляция у файла LR1_3.avi, это объясняется тем, что в видео с диктором все кадры практически идентичны, фон не меняется, не происходит каких-либо резких движений.

Оставшиеся 2 графика похожи, так происходит потому что в обоих видеофайлах есть движение, кадры отличаются друг друга. Также можно заметить резкий спад значения коэффициента корреляции в конце файла LR1_1, все потому что первый и последний кадр сильно отличаются друг от друга.

8. Алгоритм обработки последовательности видеокадров (в соответствии с индивидуальным заданием)

Индивидуальное задание: Изменение размеров исходного кадра в 2 раза по ширине и высоте

Для выполнения данного задания необходимо записывать в новый кадр четные столбцы и четные строки. В результате работы программы получаем файл **out_dop.avi**, в котором высота и ширина высота уменьшены в 2 раза.

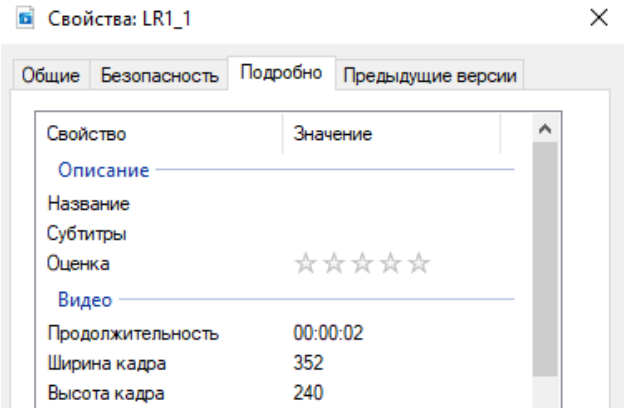

Рис. 11 – Свойства файла LR_1

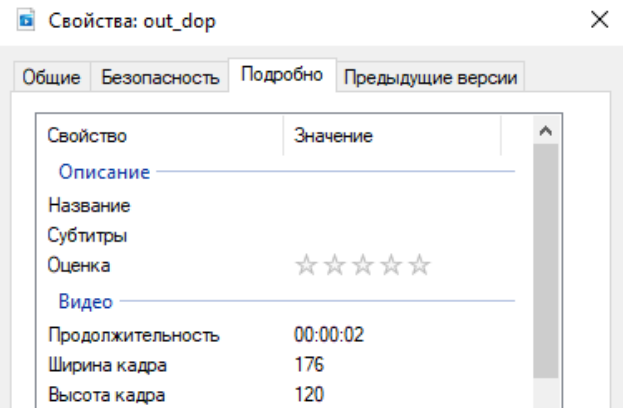

Рис. 12 – Свойства полученного файла, в результате работы кода

9. Выводы

В ходе данной лабораторной работы я ознакомился со структурой формата AVI, научился считывать, обрабатывать и работать с данным форматом, а также научился изменять размеры видеофайла. Таким образом, я получил видео с уменьшенными размерами; обратное видео и видео, состоящее из двух коротких отрывков. Также были получены графики автокорреляции, более подробно о них рассказывается в пункте №7.

10. Листинг программы

```
import av
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image

def in_video(input):
    stream = input.streams.video[0]

    W = stream.width
    H = stream.height
    format = stream.pix_fmt
    codec = stream.codec_context.name
    rate = stream.average_rate

    frames = []
    for frame in input.decode(stream):
        image = np.array(frame.to_image())
        frames.append(image)

    return stream, W, H, format, codec, rate, frames


def out_video(name_video, W, H, format, codec, rate):
    output = av.open(name_video, mode='w')

    stream_out = output.add_stream(codec, rate=rate)
    stream_out.width = W
    stream_out.height = H
    stream_out.pix_fmt = format

    return output, stream_out


def autocorrelation(frames):
    coefs = []

    signal = np.ravel(np.array(frames))
    one_frame_signal_size = len(signal) / len(frames)
    for x in range(-(len(frames) - 1), (len(frames) - 1)):
        if x < 0:
            signal_1 = signal[0:int(x * one_frame_signal_size)]
            signal_2 = signal[int(-x * one_frame_signal_size):]
        else:
            signal_1 = signal[int(x * one_frame_signal_size):]
            signal_2 = signal[0:int(-x * one_frame_signal_size)] if x != 0 else
signal

        autocor = ((signal_1 - np.mean(signal_1)) * (signal_2 -
np.mean(signal_2))).sum() / \
                        (len(signal_1) * np.std(signal_1) * np.std(signal_2))

        coefs.append(autocor)

    return coefs


input_1 = av.open('LR1_1.AVI')
stream_1, W_1, H_1, format_1, codec_1, rate_1, frames_1 = in_video(input_1)
input_2 = av.open('LR1_2.AVI')
stream_2, W_2, H_2, format_2, codec_2, rate_2, frames_2 = in_video(input_2)
input_3 = av.open('LR1_3.AVI')
stream_3, W_3, H_3, format_3, codec_3, rate_3, frames_3 = in_video(input_3)

#Correlation
```

```
plt.figure()
num = 1
for frames in [frames_1, frames_2, frames_3]:
    corr_coefs = autocorrelation(frames)
    plt.plot(list(range(-(len(frames) - 1), (len(frames) - 1))), corr_coefs,
label=('LR1_' + str(num)))
    num += 1

plt.xlabel('frame')
plt.ylabel('corr coef')
plt.grid(True)
plt.legend()
plt.show()

# Reversed
output, stream_out = out_video('reversed.AVI', W_1, H_1, format_1, codec_1,
rate_1)
tmp_frames = list(frames_1)
for i in range(len(frames_1) // 2):
    tmp = tmp_frames[i]
    tmp_frames[i] = tmp_frames[len(frames_1) - i - 1]
    tmp_frames[len(frames_1) - i - 1] = tmp

for frame in (tmp_frames):
    image = av.VideoFrame.from_image(Image.fromarray(frame, mode='RGB'))
    for packet in stream_out.encode(image):
        output.mux(packet)

output.close()

# Concatenated
output, stream_out = out_video('concat.AVI', W_1, H_1, format_1, codec_1,
rate_1)
for frame in frames_1 + frames_2:
    image = av.VideoFrame.from_image(Image.fromarray(frame, mode='RGB'))
    for packet in stream_out.encode(image):
        output.mux(packet)

output.close()

#2c
output, stream_out = out_video('out_dop.AVI', W_1/2, H_1/2, format_1, codec_1,
rate_1)
for fr in range(len(frames_1)):
    row = 0
    column = 0
    new_frame = []
    for i in range(H_1):
        if i % 2 == 0:
            new_frame.append([])
            for j in range(W_1):
                if j % 2 == 0:
                    new_frame[row].append(frames_1[fr][i][j])
                    column += 1
            row += 1
    image = av.VideoFrame.from_image(Image.fromarray(np.array(new_frame),
mode='RGB'))
    for packet in stream_out.encode(image):
        output.mux(packet)


output.close()


input_1.close()
input_2.close()
input_3.close()
```