Intelligent Signal Processing

End of Term Assignment

Exercise 1

	Total number of cars	Cars per minute
Traffic_Laramie_1.mp4	6	2
Traffic_Laramie_2.mp4	4	2

```
File Name and Duration:
Traffic_Laramie_1.mp4 2:57
Cars in 1st minute:  2
Cars in 2nd minute:  2
Cars in final minute: 2
Total number of cars: 6

File Name and Duration:
Traffic_Laramie_2.mp4 1:45
Cars in 1st minute:  2
Cars in final minute: 2
Total number of cars: 4
```

Analysis of Application

Both applications 1.1 and 1.2 utilizes the OpenCV library to analyze two traffic videos ("Traffic_Laramie_1.mp4" and "Traffic_Laramie_2.mp4"). It employs background subtraction, frame differencing, morphology operations, contour detection and my own tracking algorithm class, to identify and track moving cars, within a defined region of interest representing Main Street. Exercise 1.1 tracks all moving cars within Main Street, while Exercise 1.2 utilize the tracking class to uniquely identify each car, allowing us to count when a new car has entered Main Street's frame, that is heading towards the city centre. The tracking algorithm utilizes the mathematical concepts of Euclidean Distance and Centroids to achieve this tracking and counting. The detected and tracked cars are displayed using green rectangles.

Frame differencing and background subtraction

My code utilizes frame differencing and background subtraction techniques for detecting and tracking cars in the given camera recordings. Frame differencing is employed to highlight changes between consecutive frames, emphasizing moving objects and being able to track them. Initially, the video frames are captured and processed in a region of interest representing Main Street. The frames are converted to grayscale and then blurred using GaussianBlur to reduce noise. Subsequently, background subtraction is performed using the createBackgroundSubtractorMOG2 method, which models the background of the scene and identifies foreground objects. The resulting foreground mask highlights moving vehicles and eliminates static background elements. To further enhance object segmentation, morphological operations are applied as well, refining the shapes and providing more stable vehicle coordinates.

Contours are then extracted from the processed foreground, and the code iterates through each contour and identifies cars. A minimum contour area threshold is set to filter out small irrelevant objects like pedestrians and bicycles. Detected vehicles are outlined with bounding boxes, and the centroid of each vehicle is tracked using circles.

Exercise 2

Analysis

The application is an encoding and decoding app for uncompressed WAV files using the rice coding algorithm.

The encoding function `rice_encoder` takes a sample and a parameter `K` as input, where `K` represents the bit number, and calculates the quotient and remainder. The formula and methodology were adapted from the teachings provided in the Coursera Lab 9.005 Exercise 17. Implementing a rice encoder and decoder using Python, comprising of attaining the quotient and remainder, and generating a codeword for that sample.

The function `encoded_sample` uses the calculated remainder and bits to return the encoded sample in string.

The decoding function `rice_decoder` reverses the encoding process and reconstructs the original audio sample.

The `encode_and_decode` function combines the three aforementioned functions above. It reads the sound file in WAV format, encodes it using `rice_encoder` and `encoded_sample`, writes the encoded data to a new file with the extension "_Enc.ex2", then decodes the encoded file using `rice_decoder` and writes the decoded audio to a new WAV file with the extension "_EncDec.wav".

Additionally, the application at the end, prints the two sound files, 'Sound1.wav' and 'Sound2.wav', comparing the specifications and sizes of the original and the reconstructed audio files. The application demonstrates the implementation of rice coding, aiming to reduce or maintain the file size while preserving audio quality, as it is a lossless form of compression.

	Original Size	Rice (K=4bits)	Rice (K=2bits)	% Compression (K=4bits)	% Compression (K=2bits)
Sound1.wav	1MB	25MB	89MB	2400 %	8860 %
Sound2.wav	1MB	150MB	591MB	14900 %	59000 %

However, from the above table, we can infer that the size of the encoded file “_Enc.ex2”, is much larger when K = 2 bits. The rice coding preserves the lossless data, but the compression actually produces a large encoded file. The reconstructed WAV file are exactly the same in terms of file size and quality though. The reason for the significant difference in file sizes between K=2 and K=4 is likely due to the trade-off between the length of the codes and the compression efficiency, as well as the fact we are using strings to store the unary codes, which takes up more bits. With K=2, shorter codes are used, leading to a larger file size, while with K=4, longer codes are used, resulting in a smaller encoded file size. In summary, when K is higher, the modulus attained is higher, resulting in longer and better generation of the code in unary and binary.

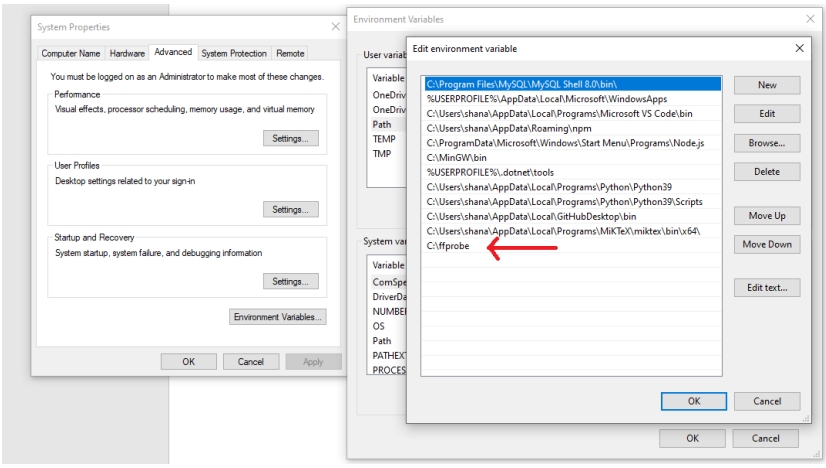
Further Development

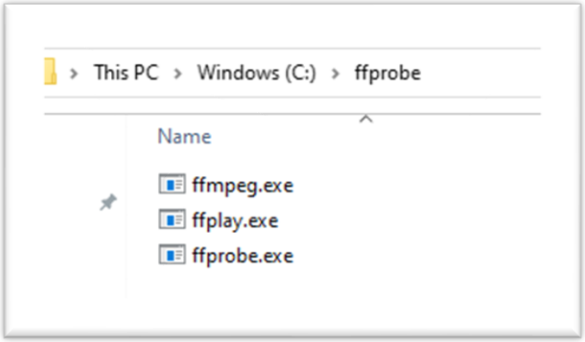
We can improve the algorithm by reducing the encoded file size. This is done by utilizing bytearrays instead of strings, as bytearrays uses less storage compared to strings.

Exercise 3

How ffprobe and ffmpeg was install and configured on my machine

I first downloaded the last full build from the official website[1]. Then added a system path variable to the `.exe` files. in my local C drive, that was obtained from the download. This allowed my system to run the ffmpeg and ffprobe commands and functionalities locally.





Analysis

The function, `generate_report`, takes the list of given video files directories as input. The function utilizes the ffprobe tool to extract metadata from each video file and checks for compliance with the specified video and audio specifications as per the assignment instructions. Using a series of built-in Python string methods and formatting, it identifies files that do not meet the required criteria, such as video codec, resolution, aspect ratio, and audio codec/bitrate, by using multiple logic statements. The function then generates a brief report in a text file, highlighting aspects of the video that do not match the requirements.

The function, `reformat_files`, takes the report generated by the previous function and the list of video file directories as input. It reads the report, identifies issues related to video and audio specifications, and attempts to reformat the problematic files accordingly. The function uses the ffmpeg tool to modify the audio codec, bit rate, sample rate, and channel, as well as the video resolution, aspect ratio, frame rate, and codec. It does this by once again using Python string formatting and logic statements. It then stores the reformatted audio and video files in the folders “videos” and “audio”, if they were reformatted, and merges them. The correctly formatted files are saved in the "reformat_videos" directory.

At the end of the notebook, the application removes all the edited video and audio files, and keeps only the original videos, and the newly re-formatted ones. This is done to save storage space.

Description of terms

- Video format (container): file format that contains both video and audio data. Common video formats include MP4, AVI, and MKV. The container holds various streams, such as video, audio, and subtitles.

- Video codec: video codec is a software tool that compresses and decompresses video data. Popular codecs include H.264, H.265 (HEVC), and VP9. They determine how video is compressed for storage and transmission, affecting file size and playback quality.

- Audio codec: audio codec is a software tool that handle the compression and decompression of audio data. Examples include AAC, MP3, and FLAC. These codecs impact audio quality and file size.

- Frame rate: the number of individual frames displayed per second in a video.

- Aspect ratio: the width and height of a video frame. Common aspect ratios include 16:9 and 4:3. Aspect ratio affects the visual presentation of the video on screens with different dimensions.

- Resolution: the number of pixels in a video display. Common resolutions include 720p and 1080p. Higher resolutions results in sharper and more vivid images/videos.

- Video bit rate: the amount of data processed per unit of time in the video stream. Measured in kilobits per second or megabits per second. The video bit rate determines the level of video compression.

- Audio bit rate: the amount of data processed per unit of time in the audio stream. Measured in kilobits per second, it indicates the level of audio compression.

- Audio channels: the number of separate audio tracks in a sound system. Common configurations include mono, 1 channel, stereo, 2 channels, and surround sound.

References

[1] ffmpeg.org. (n.d.). *Download FFmpeg*. [online] Available at: https://ffmpeg.org/download.html.