Question 1: (30 Points) ***Spatial and temporal information of videos***

In this task, you will analyze **spatial** and **temporal** information of videos. First of all, you will get in touch with the **Sobel filter**[2].

For the **spatial perceptual information (SI)**, this filter is applied to a video frame. Then, the standard deviation of each pixel in the filtered image is computed. To compute the SI of a video scene, this procedure is repeated for all frames and the maximum is chosen as SI for the scene.

The temporal perceptual information (TI) takes the motion difference between consecutive frames into account. The TI value between two frames is the standard deviation of its difference. If you want to compute the TI for a complete scene, the TI is computed for all consecutive video frames and the maximum is chosen as TI for the scene. More information can be found here: https://tinyurl.com/ya4nxmu2 (Subsections 5.3.1 and 5.3.2).

a) Download the Sintel trailer and the Big Buck Bunny clip. Convert both videos from .y4m to .yuv what is the reason for the slightly decreased file size after conversion?

Answer: the conversion is done using FFMPEG-

```
ffmpeg -threads 1 -i sintel_trailer_2k_480p24.y4m sintel.yuv -hide_banner


ffmpeg -threads 1 -i big_buck_bunny.y4m big_buck_bunny.yuv -hide_banner
```

Flags used:

- `-threads 1` defines how many threads FFMPEG will use
- `-i` defines input source
- `-hide_banner` ask ffmpeg to hide the banner with the information about the program

File data after conversion:

file name	resolution	frame rate	color space
sintel.yuv	854x480	24	yuv420
big_buck_bunny.yuv	704x480	24	yuv420

We can observe that the .y4m files are a bit bigger than the .yuv files-

file	how much the .y4m file is bigger than the .yuv file (megabytes)
sintel	7,561
big buck bunny	4,384

The reason for the slightly decreased file size is the fact that in contrast to .yuv file which is pure raw video file without any Meta data addition, .y4m contains headers and Meta data on the video (for example size, frame rate and color space)

Source: https://github.com/stoyanovgeorge/ffmpeg/wiki/Encode-Raw-Video

b) Choose any frame from the Sintel or Big Buck Bunny clip and apply the Sobel filter on its Y-component (scipy.ndimage might help).

Answer:

Extracting all frames from the sintel video using FFMPEG-

```
ffmpeg -video_size 854x480 -r 24 -pixel_format yuv420p -i sintel.yuv output-%d.png
```

Flags used: since .yuv video has no header, we need to specify the video parameters-

- `-video_size` the video resolution
- `-r` the frame rate
- `-pixel_format` the color space

Frame number 161 was chosen.

Applying Sobel filter:

See corresponding code.

c) Display the original frame, the Y-channel of the frame, and the frame with applied Sobel filter. What do you notice?

we can notice:

the information that the Sobel filter gives (e.g. edge detection) can be derived from the Y channel alone, without the need for the other chanels.

why? because the Sobel filter calculates deriative that gives the local change in **form**. the Y channel contains the image form data, and as we saw in exercise 1, its values are more intensive relative to the Cb and Cr channels, hence all the required information can be drvied from it.

d) Write your own function for spatial perceptual information measurement, called *computeSI*. The input is, upon others, the raw .yuv video. It returns SI for all frames.

Answer:

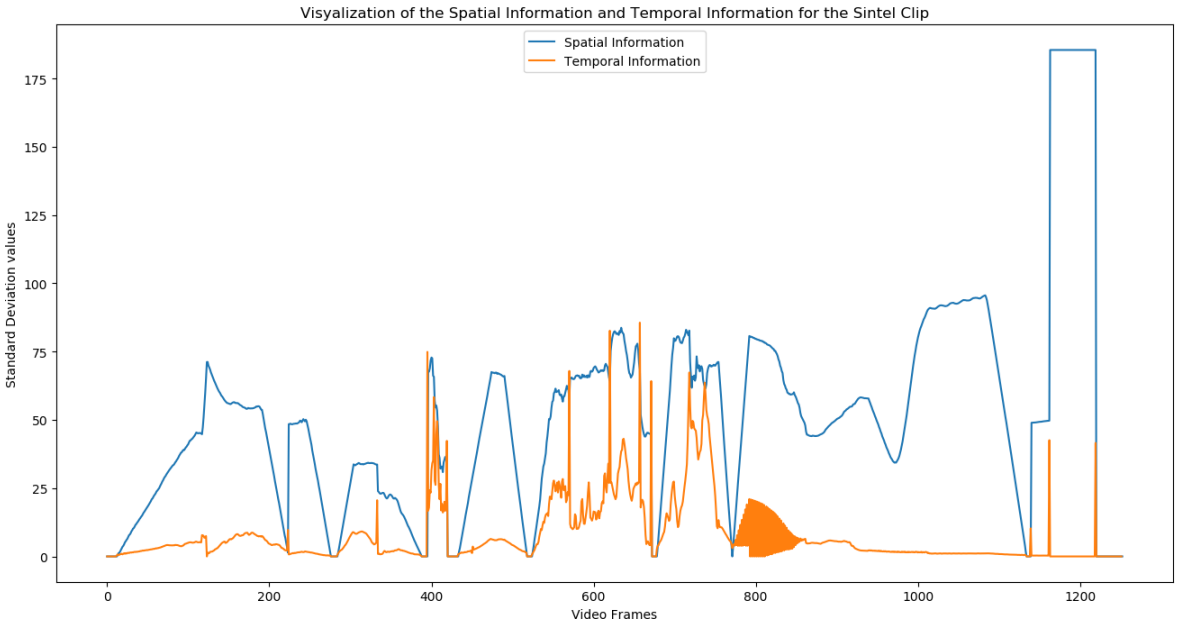
See corresponding code.

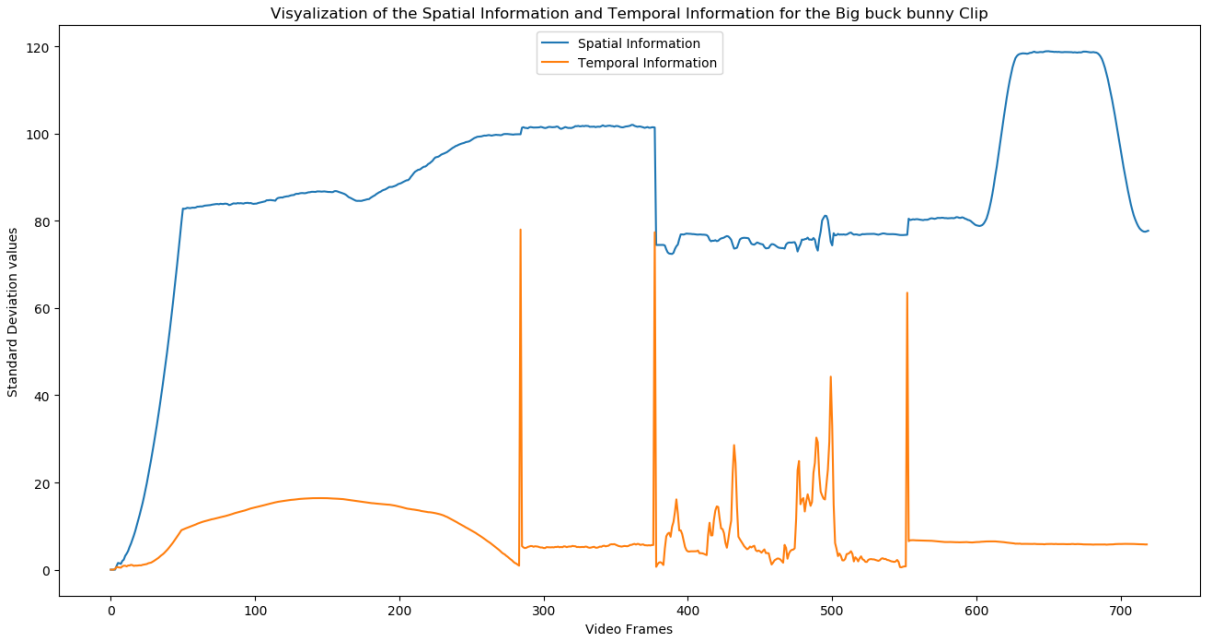
e) Write your own function for temporal perceptual information measurement, called *computeTI*. The input is, upon others, the raw .yuv video. It returns TI for all frames.

Answer:

See corresponding code.

f) Visualize the SI and TI values for all frames of Sintel and Big Buck Bunny.





Question 2: (45 Points) **Preparing videos for adaptive streaming**

HTTP Adaptive Streaming (HAS) allows to dynamically adjust the video quality to current net- work conditions. To do so, the video is split into several segments (video chunks) of equal duration, typically between two and ten seconds. Each video chunk is made available in different resolutions/bitrates, so to obtain a set of different qualities. After each video segment download, the client decides via a heuristic about the next segment's quality. In this task, you are asked to evaluate the impact of different segment durations.

(a) Use the Big Buck Bunny full movie and split the video into chunks of the following durations: seg dur = 0.5, 2, 4, 8, 12 seconds. Thereby, consider VBR encoding, with the following quality settings: crf = 18, 23, 28. Segment the videos so to obtain files of the type .ts and a playlist of the type .m3u8. Write down the ffmpeg command and all necessary flags. Also explain why the flags you used are needed and what they do.

Answer: the conversion is done using FFMPEG and the following command:

```
ffmpeg -i ../images/big_buck_bunny_480p24.y4m    -i ../images/big_buck_bunny_480p24.y4m    -i
../images/big_buck_bunny_480p24.y4m \
-threads 1 -force_key_frames "expr:gte(t,n_forced * $SEG_DUR)" \
-b:v:0 5500k -c:v h264 -profile:v main -crf:v:0 18 -map '0:v' \
-b:v:1 3500k -c:v h264 -profile:v main -crf:v:1 23 -map '1:v' \
-b:v:2 2500k -c:v h264 -profile:v main -crf:v:2 28 -map '2:v' \
-f hls -hls_time $SEG_DUR -hls_flags independent_segments -hls_list_size 0 \
-var_stream_map "v:0 v:1 v:2" \
-master_pl_name big_buck_master.m3u8 ../q2/$SEG_DUR/vs%v/big_buck_bunny.m3u8
```

Flags used:

- `-threads 1` defines how many threads FFMPEG will use
- `-i` defines input source
- `-force_key_frames "expr:gte(t,n_forced * $SEG_DUR)"` defines positions where to enforce keyframe insertion using an expression. The current setting allows to force keyframe every $SEG_DUR seconds.
- `-b:v:x 5500k` defines a desired video bitrate of the output stream number x
- `-c:v h264 -profile:v main` defines the use of H.264 video codec with a main profile of H.264 codec
- `-crf:v:x y` defines the crf quality setting of y for video stream number x
- `-map 'x:v'` creates a mapping between parameters and video stream number x
- `-f hls` enforces creation of output file according to the HTTP Live Streaming (HLS) specification
- `-hls_time $SEG_DUR` defines the target segment length in seconds ($SEG_DUR). Segment will be cut on the next keyframe after this time has passed unless you force a key frame (see above)
- `-hls_flags independent_segments` adds a flag to playlist signalizing that all the segments of that playlist are guaranteed to start with a keyframe

- `-hls_list_size 0` defines number of a playlist entries. 0 means keep all entries (by default 5, which saves only last 5 segments)
- `-var_stream_map "v:0 v:1 v:2"` defines mapping of streams and output files. Current setting means that every video stream is saved independently
- `-master_pl_name big_buck_master.m3u8` defines a master playlist name
- `../q2/$SEG_DUR/vs%v/big_buck_bunny.m3u8` defines output path. Each video stream is saved in a separate folder (%v) and for each segment duration

(b) Give some basic statistics on the resulting videos (a table is sufficient):
1. Number of video chunks
2. Total (fragmented) video size
3. Average segment size
4. Standard deviation of segment size

Answer: Depending on a segment duration we obtain the following results:

Segment duration	CRF	Number of video chunks	Total (fragmented) video size [MB]	Average segment size [KB]	Standard deviation of segment size [KB]
0.5	18	1193	188.1	161.4	82.4
	23	1193	113.5	97.4	48.0
	28	1193	67.5	57.9	27.5
2	18	299	121.6	416.6	252.1
	23	299	68.2	233.6	126.8
	28	299	38.2	130.7	62.8
4	18	150	110.2	752.0	454.4
	23	150	60.5	412.9	220.3
	28	150	33.2	226.8	104.0
8	18	75	105.1	1434.5	802.1
	23	75	57.0	778.6	380.0
	28	75	31.0	423.3	174.9
12	18	50	103.5	2120.2	1162.5
	23	50	56.0	1147.6	556.2
	28	50	30.4	623.2	254.6

(c) Take a look on the playlist (you can easily open it with any text editor). Which information does it provide to you?

Answer:

First let us consider the master playlist. It is created once for a video and contains information about different video/audio streams available. Here are the contents of master playlist for three video streams of different quality (crf = 18, 23, 28):

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-STREAM-INF:BANDWIDTH=6050000,RESOLUTION=704x480,CODECS="avc1.4d401e"
vs0/big_buck_bunny.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=3850000,RESOLUTION=704x480,CODECS="avc1.4d401e"
vs1/big_buck_bunny.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=2750000,RESOLUTION=704x480,CODECS="avc1.4d401e"
vs2/big_buck_bunny.m3u8
```

EXTM3U – the first line is the header which signalizes that this is an extended M3U file.
EXT-X-VERSION – the second line defines the version of M3U format.
EXT-X-STREAM-INF – the third line indicates that the next URL in the playlist file identifies individual stream playlist file. The following parameters in third line define additional information about defined video stream:

BANDWIDTH – an integer that is the upper bound of the overall bitrate for each media file, in bits per second

RESOLUTION – optional display size, in pixels, at which to display all of the video in the playlist

CODECS – audio/video codecs declaration according to RFC 6381. For example, avc1.4d401e as above defines H.264 codec with Main Profile Level 3

Secondly let us investigate the **playlist for each separate stream.** Here are the contents of playlist for 0.5 segment duration, crf = 18 quality video:

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:0
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-INDEPENDENT-SEGMENTS
#EXTINF:0.500000,
big_buck_bunny0.ts
#EXTINF:0.500000,
big_buck_bunny1.ts
. . .
#EXTINF:0.458333,
big_buck_bunny1192.ts
#EXT-X-ENDLIST
```

EXT-X-TARGETDURATION – specifies the maximum media-file duration. The value of 0 as shown above might be a bug in ffmpeg. Since this should be an integer value, but every segment is smaller than 1.
EXT-X-MEDIA-SEQUENCE – indicates the sequence number of the first URL that appears in a playlist file. Each media file URL in a playlist has a unique integer sequence number. The sequence number of a URL is higher by 1 than the sequence number of the URL that preceded it
EXT-X-INDEPENDENT-SEGMENTS – indicates that every segment is playable independently and guaranteed to start with a keyframe
EXT-INF – a record marker that describes the media file identified by the URL that follows it. Each media file URL must be preceded by an EXTINF tag. This tag contains a duration attribute that's an integer or floating-point number in decimal positional notation that specifies the duration of the media segment in seconds. This value must be less than or equal to the target duration

(d) Have a closer on the frames' characteristics. For this task, only consider the fragmented videos resulting from the encoding with crf = 23.

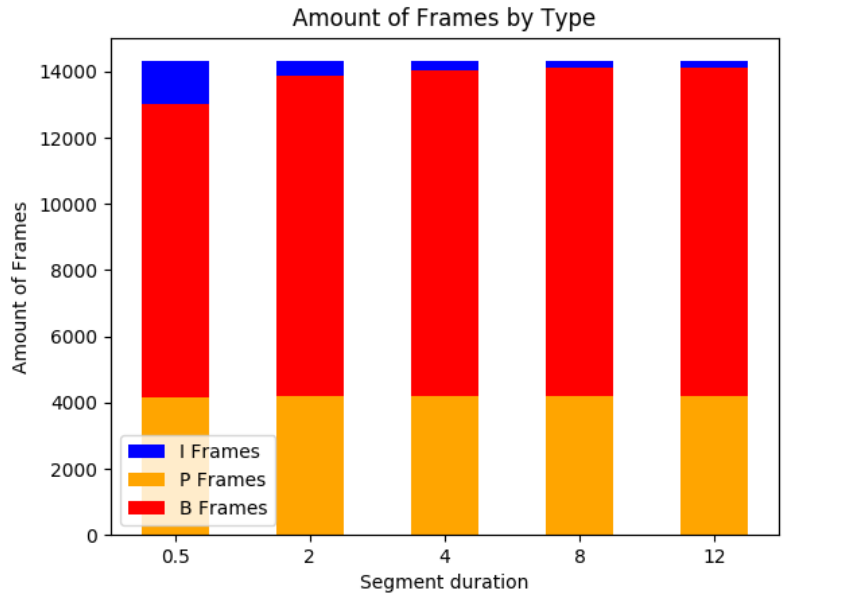
1. How many I-frames in total are required for the different segmentations, i.e. seg dur = 0.5, 2, 4, 8, 12?
2. How much do the I-frames contribute to the overall file size in the different segmentations?

Answer:

1. We obtained the following results of total I frames count for crf = 23 video chunks:

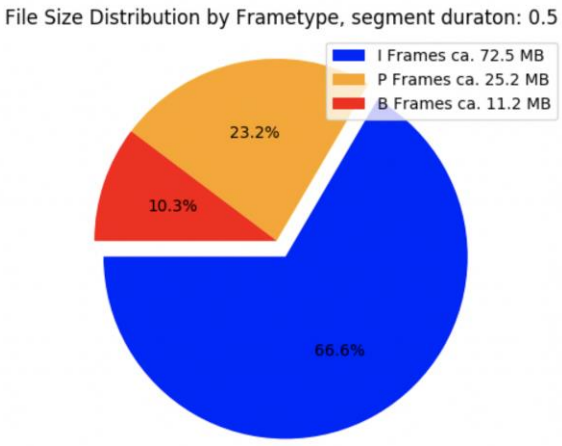
Segment duration	I frames count	P frames count	B frames count
0.5	1310	4147	8858
2	428	4194	9693
4	281	4192	9842
8	209	4200	9906
12	192	4197	9926

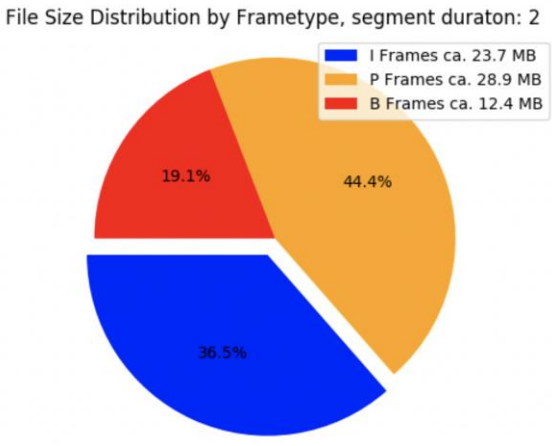
The visualization of frames amount by type is the following:

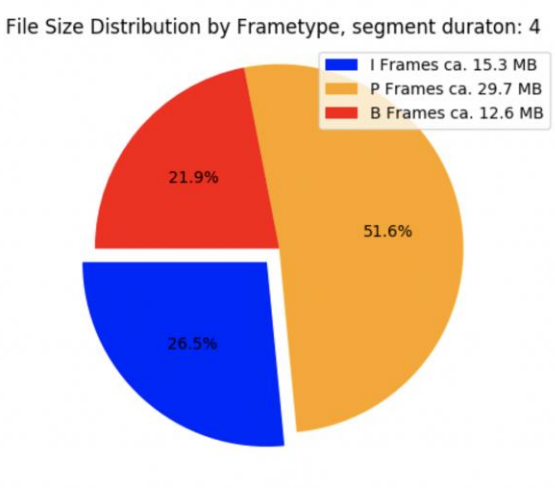


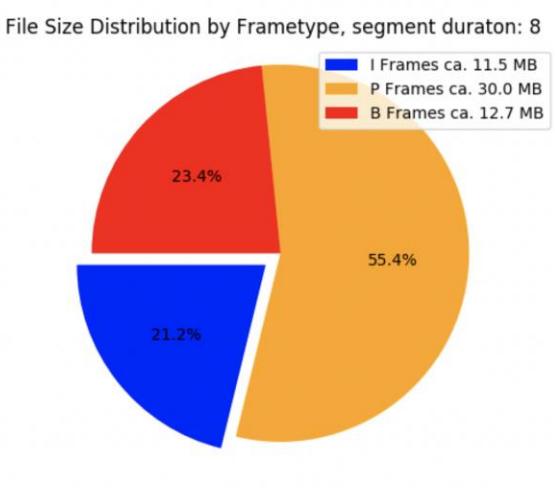
Noticeably that total frames amount does not change, as it is always 14315.

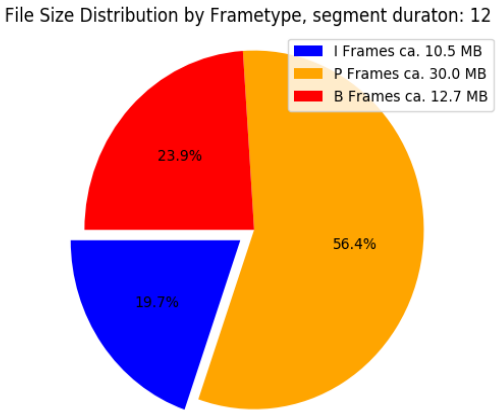
2. As we are forced to insert at least one I frame in every single chunk at the beginning (in order to make it playable independently), amount of I frames is dependable on the segment length we are going to choose. It starts getting problematic when segment duration is smaller than the duration at which we would not need an I frame yet, if we were encoding the whole video on a continuous manner.
   Thus, the high compression is achieved due to an intensive usage of B frames, as the amount of P frames did not change, it is clear that only some part of I frames was substituted through B frames. The following pie charts we obtained for different segment durations:











We can see that as the number of I frames decreases and number of B frames increases (from 'Amount of Frames by Type' chart), the data consuming I frames are substituted through lightweight B frames, which makes the encoding process more efficient.

(e) Have a look on the average SSIM of the videos. Does it notably change among the different segmentations?

Answer:

SSIM values were obtained with the following ffmpeg command:

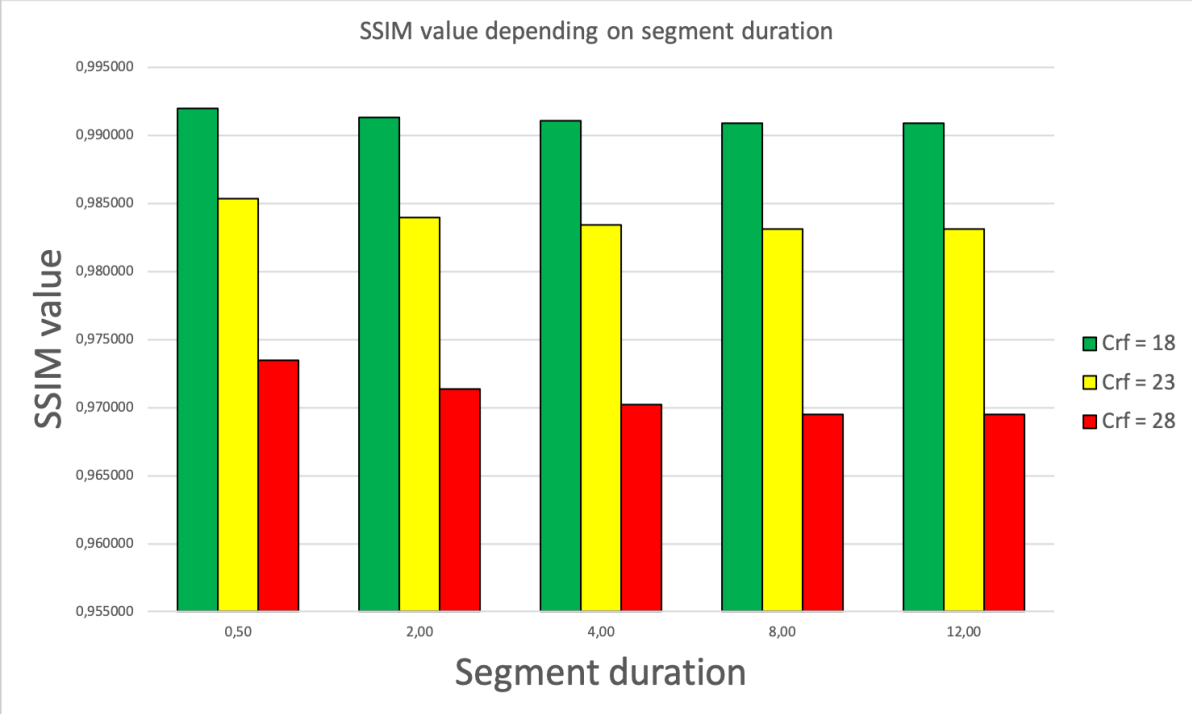
```
ffmpeg -threads 1 \
-i ../q2/$SEG_DUR/vs0/big_buck_bunny.m3u8 \
-i ../images/big_buck_bunny_480p24.y4m    \
-lavfi    ssim=../q2/SSIM/ssim_18_$SEG_DUR.log -f null -
```

Flags used:

- `-threads 1` defines how many threads FFMPEG will use

- -lavfi defines Libavfilter input virtual device
- -ssim defines SSIM metrics to be calculated and log file name

Results are visualized on the following chart:



The first thing to mention is that SSIM values overall are almost always higher than 0.97. Indeed, video quality is overall very satisfying.

Depending on crf, SSIM values are changing more or less noticeably:

1. Due to an increased use of I frames in 0.5 duration segments, SSIM values are higher there, than for any other segment duration.
2. For a visually lossless crf value of 18, SSIM value seems to be almost not affected at all, whereas for crf of 28 codec shows a worse SSIM value for a longer segment duration.

(f) Considering results you obtained in this task, would you rather take longer or shorter video segments?

Answer:

Longer video segments reducing file size and overhead on the server side and on the client side, shorter video segments improving parameters like random search and time of quality change.

Hence, it depends:

For a VoD streaming service we would suggest a segment size of around 2-3 seconds, as it's a good tradeoff between quality of experience and data efficiency. Segment durations of 8 and 12 seconds would not deliver a drastic change in codec efficiency but would hardly affect the overall experience by causing a stalling. However, if the computation resources are limited, longer segments should be considered.

(g) In the context of adaptive video streaming, what are the benefits and drawbacks of long video segments and small video segments?

Answer:

Smaller video segments:

Benefits-

1. ***Random video access:*** robust and fine-grained random video access. When user wants to jump into a different section of video, he does not have to experience stalling before video starts playing again. This allows quick search for a specific moment in the video.
2. ***Quality changes:*** If user had a low bandwidth when the video had start playback, and so got a lower quality segment, he may achieve the higher quality experience after the first segment quicker. Since the segments are smaller.

Drawbacks-

File size: larger file size due to increased number of required I frames. Which affect storage on the server, consume more bandwidth on average, and required more resources on client side, including more calculation regarding which segment to ask next.

Larger video segments:

Benefits-

File size: smaller file size, due to more efficient encoding. Which affect storage on the server, consume less bandwidth on average, and required less resources on client side.

Drawbacks-

1. ***Random video access:*** Slow random access to the video. User has to wait more while the required segment is fetched.
2. ***Quality changes:*** Slow quality adjustment. User has to wait longer time (since the segment duration is longer) for a change in quality.

[Source: course slides]

Question 3: (10 Points) ***Introduction to video streaming***

Download the file ConventionalController.py (https://service.inet.tu-berlin.de/owncloud/index.php/s/jr0HQulSwhxGtQs) and the file main playlist.m3u8 (https://service.inet.tu-berlin.de/owncloud/index.php/s/by3TGttGQxQ6xqd).

(a) Have a look on the file called ConventionalController.py. It is an implementation of an HAS heuristic. Find out which information is taken into account when deciding about the next segment's quality. In this context, also describe the purpose of the following functions:
1. calcControlAction
2. quantizeRate
3. ewma filter

Answer:
Information taken into account when deciding about the next segment's quality:

- available video segments and their quality
- estimation of the bandwidth that will be available for downloading the next segment (based on past downloads)
- the current buffer length

1. calcControlAction
Purpose: determine and returns the estimated available data rate (bytes/second) for fetching next segment. This rate is passed to the method *quantizeRate.*

2. quantizeRate
Based on the rate given from *calcControlAction*, *quantizeRate* determine in which quality should the next segment be downloaded (quality in quantization rate)

3. ewma filter
ewma (*'exponential weighted moving average')* is called by *calcControlAction* in order to calculates the estimated available data rate for fetching next segment, and it does it by weighting past bandwidth samples.

(b) Now take a look on the file main playlist.m3u8. It is typically located on the video server and downloaded by the video client. Which information does it provide to the video client?

Answer:
the information provided is-

- Type of playlist
- playlist ID
- Information about individual stream playlist, and in particular it's required bandwidth

[Source: Conferences2.sigcomm.org. (2019). [online] Available at: http://conferences2.sigcomm.org/co-next/2014/Workshops/VideoNext/VideoNEXT_papers/p1.pdf [Accessed 13 Jan. 2019].]

Answers for Question 4**:**

4(a):

1. own play list: the play list “big_buck_master.m3u8” used located in folder *“misc”*
2. videos available on the server: Done-

```
vagrant@vagrant:/var/www/html$ ls
content  index.html  test-content
vagrant@vagrant:/var/www/html$ la content/
total 48K
660439 4.0K drwxr-xr-x 5 root root 4.0K Jan 14 08:52 .
664697 4.0K drwxr-xr-x 4 root root 4.0K Jan 14 08:52 ..
665071 4.0K -rw-r--r-- 1 root root  329 Jan 14 08:52 big_buck_master.m3u8
664855  12K drwxr-xr-x 2 root root  12K Jan 14 08:52 vs0
665072  12K drwxr-xr-x 2 root root  12K Jan 14 08:52 vs1
660461  12K drwxr-xr-x 2 root root  12K Jan 14 08:52 vs2
vagrant@vagrant:/var/www/html$ cat content/big_buck_master.m3u8
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-STREAM-INF:BANDWIDTH=2750000,RESOLUTION=704x480,CODECS="avc1.4d401e"
vs2/big_buck_bunny.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=3850000,RESOLUTION=704x480,CODECS="avc1.4d401e"
vs1/big_buck_bunny.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=6050000,RESOLUTION=704x480,CODECS="avc1.4d401e"
vs0/big_buck_bunny.m3u8


vagrant@vagrant:/var/www/html$ cat content/vs
vs0/ vs1/ vs2/
vagrant@vagrant:/var/www/html$ cat content/vs0/big_buck_bunny
Display all 151 possibilities? (y or n)
vagrant@vagrant:/var/www/html$ cat content/vs0/big_buck_bunny.m3u8
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXT-X-TARGETDURATION:4
#EXT-X-INDEPENDENT-SEGMENTS
#EXTINF:4.000000,
big_buck_bunny0.ts
#EXTINF:4.000000,
big_buck_bunny1.ts
#EXTINF:4.000000,
```

3. re-provision VM: Done

4. run command with own playlist: Done
5. everything worked: Done, please see logs in folder *"misc"*

4(b):

The information that can be gathered from the log file:

- **"ts"**- absolute time stamp (in unix time)
- **"enqueued_b"** - buffered bytes available in queue
- **"enqueued_t"** - buffered time available in the queue
- **"bwe"**- connection bandwidth estimation
- **"cur"**- current quality in bitrate
- **"level"**- current segment quality level
- **"max_level"**- maximum available quality
- **"player_status"**- indicates if the player has started streaming (0 = no, 1 = yes)

4(c):

Do measurements with different settings of the available bandwidth: Done, please see logs in folder *"misc"*

4(d):

Visualize how buffered play time over time for the four bandwidth configurations: Please see figure below. The graph visualize the correlation between available bandwidth, and available buffer size over time.

For all bandwidths: The reason to the small spikes in the graph is that we fetching new segment only after a segment is consumed. Also, we do not observe the final stage of empting the buffer since the system is configured to stop when there is no more segment to fetch.

Bandwidth configured to be as the average bitrate of the crf = 28 segments: the low bandwidth force the client to fetch almost only the lowest quality segments, and still the buffer is very low, and the video is stalling. The bandwidth is not enough in order to fetch the segments quick enough.
We see initial buffer build up, when the first segments are downloaded, and then the low bandwidth cause the buffer to get empty before it could be re-filled. The big spikes are due to variant segments size which cause the buffer filling time to increase.

Bandwidth configured to be as the average bitrate of the crf = 23 segments: the client tries to optimize the quality, and it cause reduction in the buffer size. After the initial build up, the buffer spikes in correlation to the segment size, as can be seen ca. second 410, were the complex bird scene cause a selection of lower quality and buffer drain.

Bandwidth configured to be as the average bitrate of the crf = 18 segments: the client tries to optimize the quality, but also manage to keep high buffer size. After the initial build up, the buffer spikes in correlation to the segment size, as can be seen ca. second 410, were the complex bird scene cause a selection of lower quality and buffer drain.
The changes between configuring the bandwidth to be as the average bitrate of the crf = 18 segments, and as the double of the average bitrate of the crf = 18 segments is minor. In both cases we have enough bandwidth to download optimized segments for both quality and buffer size.

4(e):

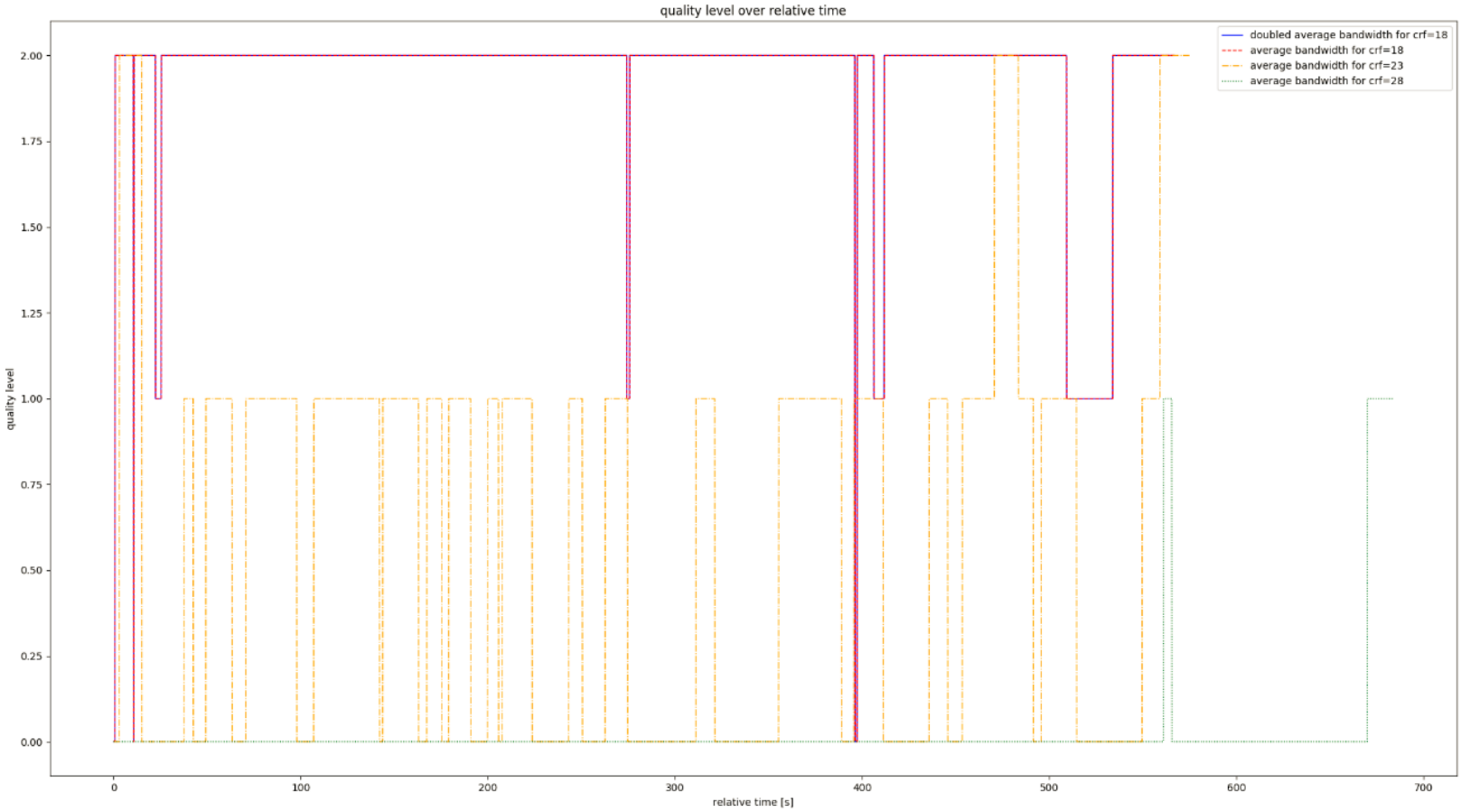
Visualize the quality level over time for the four bandwidth configurations: Please see figure below. The graph visualize the correlation between available bandwidth, and selected segment quality over time.

For all bandwidths: during the slow start, the lowest quality segments are selected in order to fill up the buffer. Complex scenes force the selection of lower quality segments.

Bandwidth configured to be as the average bitrate of the crf = 28 segments: the low bandwidth force a selection of the lowest quality almost always. The longest download duration is due to stalling.

Bandwidth configured to be as the average bitrate of the crf = 23 segments: the client tries to optimize the quality, and it manage to fetch many mid-level quality segments, and some best level segments. the slightest longer download time is due to the slow start in comparison to the highest bandwidth.

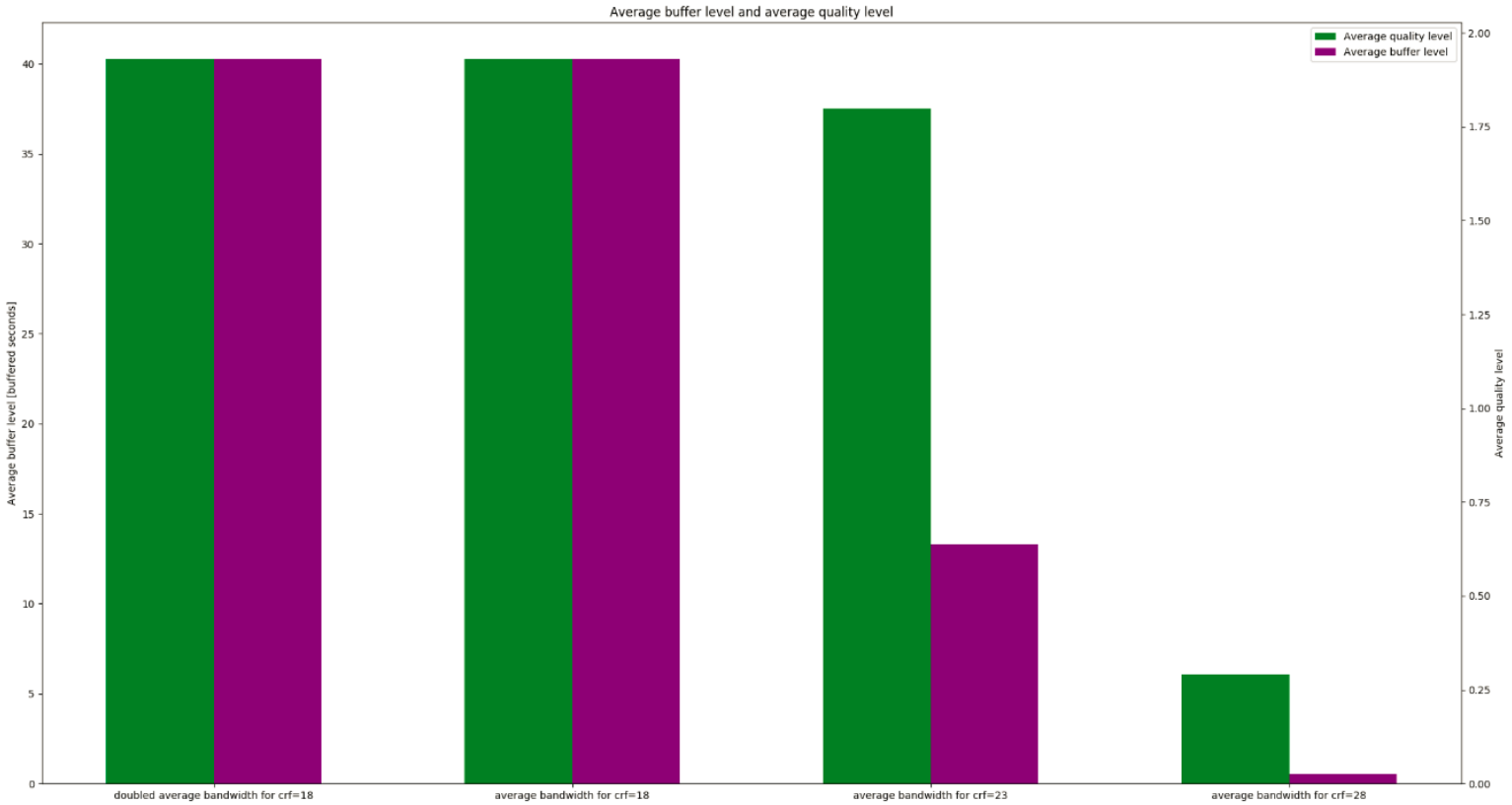
Bandwidth configured to be as the average bitrate of the crf = 18 segments: due to low bandwidth, the client is forced to select almost only worst quality segments.



4(f):

Visualize the average buffer level and the average quality for all bandwidth configurations: Please see figure below. The graph visualize the correlation between average quality and available buffer.

we can observe the trade-off thet enforced by the bandwidth limitation between average quality and



available buffer size.

we also can see that above some threshhold, adding bandwidth does not improve the quality or the available buffer (bandwith=av. bitrate for cef=18 and double the bandwith gives the same results) and that below some threshold, even though theoreticlly we suppose to have enough bandwith, both the quality and the buffer available are dropping.