05/25/2018

REPORT ON WHY FPS OF A LIVE STREAM IS NOT CONSTANT AND HOW FPS ON A STREAM RECORDING IS SET

Basics Understanding and flow of streaming from web url on python script

Before one wishes to gain an understanding of how the frames per second (fps) on a live stream affects the duration of a stream recording, one should gain a better understanding of the components involved in the streaming process.

The components involved in streaming of a web camera live stream can be divided into three general groups: capturing frames, transmission and streaming.

The capturing frames component includes the camera that is used to capture live footage. Every camera has a fps setting. Every frame is a still image captured and the fps represents the number of frames the camera is set to capture per second. A high fps means the camera is capable of capturing more still images per second.

The transmission component includes communication mechanisms that serve as a medium for the data (frames) to be transferred.

The streaming component includes the program used to read the frames from the transmission component. Using our python script as an example, the streaming process is as follows:

1. Reading the frames using OpenCV’s VideoCapture object/ why the fps of stream is different from the fps set on a camera

    The continuous frames are read using VideoCapture.read() using a while loop. One should be aware that VideoCapture.read() is a blocking function that context switches the current process out when a frame from the transmission component is not available (empty). Hypothetically, owing to the transmission component overhead, frames do not arrive at a stable rate. Therefore, when a frame does not arrive in time to be read, VideoCapture.read() context switches out in the absence of a frame and only context switches back to the process when a frame arrives.

    One should also be aware that each frame is read per iteration of the while loop. This means that the *frames that can be read is dependent on the speed of the while loop used*. We should be know by now that a recorded video (mp4,avi) with its own playback duration have a constant number of frames while the number of frames when streaming increases with time. Therefore, when we apply this concept to a recorded video (mp4,avi) instead of a stream, even though a recorded video should have a constant fps, doing read() in a continuous loop on the recorded video (mp4,avi) might be faster or slower depending on how fast each read is done because (*ad nauseam*) *it is dependent on the speed of the while loop iteration*.

These two factors explain why the frames read per unit time (second) differs throughout the stream and is not a constant. The frames read per unit time translates to the fps of the stream (not fps of the camera). Therefore, one can conclude that the fps of the "capturing frames" component does not equate to the fps of the stream.

2. <u>Writing the frames using OpenCV's VideoWriter object</u>

This python script uses VideoWriter.write(frame) to write the frame into an avi/mp4 file. In the same while loop as read(), after a frame is read, write(frame) is used to write the frame into the file.

However, in order to use write(frame) one has to create a VideoWriter object and a particular argument required is the fps. In order to illustrate why an fps is required, the VideoWriter objects takes an fps of x as argument and thus, for every x frames stores them in a one second window. Therefore, when an fps of 20 is set, if 6 frames are read in a one second window, when write is used on these 6 frames, they will not occupy a one second window but is translated to 6 frames of 20 frames of a one second window. Similarly, if 25 frames are read in a one second window, only 20 frames will be occupy a one second window and the next 5 will occupy 5 of the next 20 frames of a one second window.

Given a stream with fps that varies, one should realise by now that deciding on an fps is no trivial matter. It is imperative that the programmer inputs an fps that is appropriate that yields a recording duration as specified by the user. However, putting a constant fps will result in an avi/mp4 file that results in playback that alternates in playback speed which is unsightly to be viewed.

3. <u>Deciding on an FPS for Videowriter object</u>

A one minute stream is used to calculate the fps per second of the stream. Two values were obtained: the average fps and the mode of the fps obtained. In deciding which FPS to use, the average fps is used for writing of duration for 1 minute or less and the mode is used for writing of duration for more than 1 minute.

4. A computationally expensive, maybe computationally feasible way but definitely more accurate way of writing to get a video recording

While streaming the video, calculate the fps per second of the stream and save the fps per second in a list. At the same time, you save the stream in a recording of a constant fps. Because the end output is just an avi/mp4 with a fixed number of frames, this solution uses the fact that the the number of frames is fixed to post-process.

During post-production, the user needs to have ffmpeg or some sort of library that can combine two videos together. Then create new videowriter object for every second of the stream and create a one second video and combine with the rest of the 1 second clip.

If one second is computationally infeasible, we can increase the duration to 10 seconds. We get the fps of this 10 second block (average of 10 one second blocks) and then set the fps of the video writer.

This solution requires post production time. Because processing the video during streaming might incur overhead during the stream and slow down the streaming process, processing after the streaming is done is a better option.