Lab2: Video Compression + Distribution

In this project, you will build a video compression/coding system, examine its compression efficiency and its robustness against packet losses and errors during distribution (over the Internet or wireless networks). Each project team will include 2 students. Find your partner now!

You can use either matlab or C/C++ although matlab is preferred. To get any credit, your code must run on csil machines.

You system needs to perform the following functions:

[5%] Step 1: Video input
Read the content of a YUV based .avi file (you will be given the foreman.avi as an sample). We are only interested in the luminance component of the video. Note that foreman.avi only contains the luminance component.

[40%] Step 2: Video compression
Compress the video using prediction (or motion compensation) + DCT + quantization + run-length coding + fixed-length binary coding. Note that this run-length coding is **NOT** the coding used in your HW2 programming assignment #16. Refer to the JPEG coding lecture (Lecture 5) for details on run-length coding.

Your system must support two compression modes, and one optional mode:

[15%] Mode 1: All the video frames are coded as I-frames
[25%] Mode 2: The first of each 10 frames is coded as I-frames, and the rest as P-frames
[15%] Mode 3 (optional): The first of every 10 frames is coded as I-frames, and the rest as P, or B-frames; **Teams that implement B-frames will obtain 15% extra credit**. Note: **you need to build a successful decoder in step 3 to obtain this 15% extra credit.**

For Mode 1: you need to perform a **JPEG-like** compression on each frame: divide the frame into multiple 8x8 blocks, run DCT on each block, followed by quantization (using the luminance quantization table **Q** provided in the lecture slides) and zigzag ordering and run-length coding (except DC, for DC you use predictive coding). You will need to design your own **fixed-length** binary coding dictionary to represent the run-length code, i.e. determining the binary sequence used to represent each possible run-length code and the DC code.

For quantization (required) : you will need to support a quantization scale factor k, which is specified by input. Instead of using **Q** as the quantization step, you will need to use **kQ**, which scales each element of Q by a factor k

(k=1, 2, 3, 4…) This way you can control the amount of zeros produced by the quantization, and thus the compression ratio.

We have provided a sample file (lab2-support.zip) that implements simple DCT + simple quantization; you can use/modify this file for your own system. You can access the sample file at:
http://www.cs.ucsb.edu/~htzheng/teach/cs182/schedule/pdf/lab2-support.zip

[10%] Optional: Huffman coding -- you can also implement the compressor using the Huffman coding rather than fixed-length coding. **Teams that implement Huffman coding (and its decoding in Step 3) will obtain 10% extra credit.**

For Mode 2: you perform the same procedure on I-frames; and perform motion-estimation + compensation for P-frames. Note that for simplicity, the motion estimation/compensation is performed on 8x8 blocks. For P-frames, you code the difference after motion compensation using the above DCT + quantization + zigzag ordering + run-length + fixed-length (or Huffman) coding. You will also need to code the motion vector..

For Mode 3: you will choose whether a frame should be B or P frames and use different motion-estimation/compensation mechanisms. You will also need to code the motion vector(s).

Important: Adding Headers: Note that your system will convert a set of video frames into a long sequence of binary bits (0,1). You will need to add header information to the start of the sequence to cover the size of the video frame, the k value. You will also need to insert header or special sequence before the bit sequence of each 8x8 block, so that your decoder can recognize each 8x8 block.

[25%] Step 3: Video Decoding + Display

Read the binary sequence of your compressed video, reverse the compression steps and recover the video .avi. In this step, there is no error/loss in the compressed binary sequence. **You need to make sure that we can use matlab aviread to open your recovered .avi file and play the video.**

In this step, you will also compute the total distortion introduced by your video compression by computing the average PSNR value (the sum of PSNR value of each frame, divided by the number of frames), and the worst per-frame PSNR value (the minimum PSNR value among all the frames). Note that you need to compute PSNR by comparing the original and the recovered video frames one by one.

In this step, you will also use your video compressor to compress the video using different k, compute the total bits of the compressed video as well as the average PSNR value as the video quality. This will produce different rate (R) and average PSNR (D). You will plot a R/D curve where y-axis is the bit size and the x-axis is the average PSNR value.

[10%] Mode 1 decoding + R/D curve
[15%] Mode 2 decoding + R/D curve

Note to get the 15% in Step 2, Mode 3, you need to build a successful decoder

[30%] Step 4: Video Distribution + Robust Decoding + Display

In the previous step you design your decoder assuming there is no error/loss when distributing the compressed video content. In this step you will introduce these errors/losses and examine the performance of your decoder. You will improve your decoder (and possibly the encoder) to minimize the impact of error/loss.

You will inject packet losses/errors using the following procedure:

1) packetize the binary sequence into packets, each of 500 bytes in size. Each has a packet number;
2) You will inject (random) packet losses and errors into these packets. A packet loss means that the packet will not reach the decoder; packet errors mean that certain bits in the packet are flipped, 1→ 0, 0→1 .

A sample loss injection can be as follows:

Let's say in 1) you've produced N packets;
You can produce x% random packet loss using matlab:
losspattern=floor(rand(N*x,1)*N);
where losspattern contains the sequence number of packets which are lost.
You will remove these packets before you decode the sequence.

The typical x values we will be using to test your program are 1%, 5%, 10%, 30%

Similarly, you can selectively inject random errors in selected packets by flipping some bits. For example, we will select y=10% of packets, and flip y=10% of randomly selected bits inside each selected packet.

[10%] After passing your binary compressed video through packet losses and errors, you will attempt to use the same decoder to decode your video. **You will again compute the average and minimize PSNR values and plot it against the data rate, for x=2%, 5%, 10%, 30%. You will also include one**

where you select 10% of packets randomly and randomly inject 10% of bit errors inside each selected packet.

[15%] Now you will seek to improve your decoder (or even encoder) to minimize the impact of packet error/loss. You can search the Web for error resilience or error recovery in video; You should also consider revise your compression to make it error resilient. Write a short description on your proposed mechanisms.

Due Dates

Since this is a fairly complex project, we will use multiple checkpoints:

- **Due Tue May 11, 11:59PM via TURNIN (use lab2a in your turnin command)**

Step 1 + Step 2 (Mode 1, Mode 2) + Step 3, including the R/D curves

- **Due Tue May 18, 11:59PM via TURNIN (use lab2b in your turnin command)**

Step 4 + optional components of Step 2/3 (Mode 3 or Huffman coding).

Note that for Huffman coding you have two ways to obtain the full credit + extra 10% credit:
Option 1: implement Huffman coding and submit by May 11
Option 2: implement fixed-length coding and submit by May 11, and implement Huffman coding and submit by May 18.

This also means that by May 11, you need a working version of the encoder/decoder regardless of whether you use fixed-length or Huffman coding.

Accepted file format

The following lists the set of files that you need to submit to obtain any credit:

The System that is due May 11:

If you use matlab, you will submit three functions:

1) **vencoder.m** which defines the following encoder function**:**

R=vencoder(inputvideoname,outputfilename, mode, k)

where inputvideoname is the .avi you wish to encode/compression, outputfilename is the file to save the binary sequence, mode can be 1, 2, 3 which maps to mode 1,2, 3 described in Step 2, k is the quantization scaling factor; return R is the the bit size of the binary sequence.

2) **vdecoder.m** which defines the following decoder function:

vdecoder(inputfilename,outputvideoname, mode, k)

where inputfilename is the compressed binary sequence you wish to decode, and outputvideoname is the **.avi** filename to save the decompressed video.

3) **vpsnr.m** which defines the following PSNR computing function:
[meanp, minp]=vpsnr(orgvideo,recvideo,nFrames)

where orgvideo is the filename of the original video, recvideo is your decompressed video and nFrames is the total number of frames; it returns the meanp the average PSNR of all the frames, and the minp, the minimize PSNR across all the frames.

4) veval.m which defines the following evaluation function

[R,D]=veval(orgvideo,kmax)

where orgvideo is the original video .avi filename, kmax is the maximum value of k. This function will produce the R/D curve for k=1…kmax, where R is the compressed video size, and D is the meanp computed in vpsnr.

If you use c/c++, you will provide these functions, use the same parameters as your program input.

The System that is due May 18:

You will include two additional functions

5) **distribution.m** which defines a function that introduces packet losses/errors into your compressed video.

distribution(inputfilename,outputfilename, lossrate,errorrate)

where inputfilename is the binary compressed video content, and outputfilename is the content after it is being distributed (after packet losses and errors), lossrate represents x, and errorrate represent y described in step 4.

6) **newvdecoder.m** which defines the new video decoder that seeks to minimize the impact of losses and errors. It will have the same input/output format lat vdecoder.m

Your Demo

In your demo, you will build a script that uses veval.m to evaluate your encoder/decoder and plot the R/D curve with and without packet loss/error. You will need to prepare a 6 to 8-page presentation to explain your design. We will also use additional videos to check your program, as well as random error/loss injections.

The Demo date will be determined; and will be after May 18th.