Assignment - 1

Fundamentals of Digital Video Compression

ECE1783 - Design Tradeoffs in Digital Systems

By: Ritam Haldar (1004692950)
Felipe Sander Pereira Clark (1003911546)
Agasti Kishor Dukare (1005602019)

Due Date: October 23, 2019
Submission Date: October 23, 2019

Contents

List of Figures

1 Introduction

Video encoding/decoding is a technology that is employed when there is a need to convert a sequence of video frames from one representation domain to another, normally for efficient transmission, storage and reproducibility in different devices. In this report a detailed dissection and analysis of two video encoder/decoders with different purposes will be presented. During this analysis the relevance of parameters and their significance will be detailed. Results will then be explored through graphs and frame plots.

The first encoder/decoder - from now on referred to as "Simple Encoder" - is a simplified type, where only basic pixel value approximation is applied. The second - from now on referred to as "Elaborate Encoder" - is a more realistic one, where quantization of blocks in the discrete cosine transform domain is performed, followed by entropy encoding, gearing it towards lossy file compression.

Throughout the process the influence of several parameter will be discussed under the perspective of the final video quality, bitrate, output file size and performance. In addition, where pertinent, the relevance of intermediate steps towards the composition of the final encoded video will be highlighted.

All the software implementation for the proposed encoders/decoders was done using the Python programming language. Specific challenges and simplifications caused by the choice of this language are highlighted in Section 4.

Finally, section 5 presents final remarks, conclusions and ideas for further development.

2 The Simple Encoder

Fig. 1 shows the structure of the Simple Encoder. It has two main configurable blocks, namely approximation and prediction.

Initially, each frame of video is split into block of size $i \times i$. The block size (i) is variable and should directly affects the output, since all the major operations are block-wise. Supposedly, the higher the block size, the poorer the video quality[1], and vice-versa.

The approximation parameter $n \in [1, 2, 3]$ is used to round residual values to the nearest multiple of 2^n. For example, a pixel value $p = 13$ would be rounded to either $p_r = 12$ or $p_r = 14$ for $n = 1$, and it would be rounded to $p_r = 16$ for $n = 3$. Clearly the modulus of the approximation error is potentially greater as n increases, which sets the expectation that the higher this value, the lowest the quality of the encoded frames.

The prediction component takes $i \times i$ segmented blocks from the original video and searches for the closest block (by minimization of a cost function) in the reconstructed video in order to achieve minimum residual error. The search space is delimited by the parameter r, which sets the vertical and horizontal search spaces. It is expected that with larger r the chances of finding strongly correlated blocks are enlarged, thus leading to smaller residues and ultimately better image quality.

[1]The term "Video quality" will be generally applied to refer to subjective/perceptual quality. When objective metrics are applied, this will be made explicit.

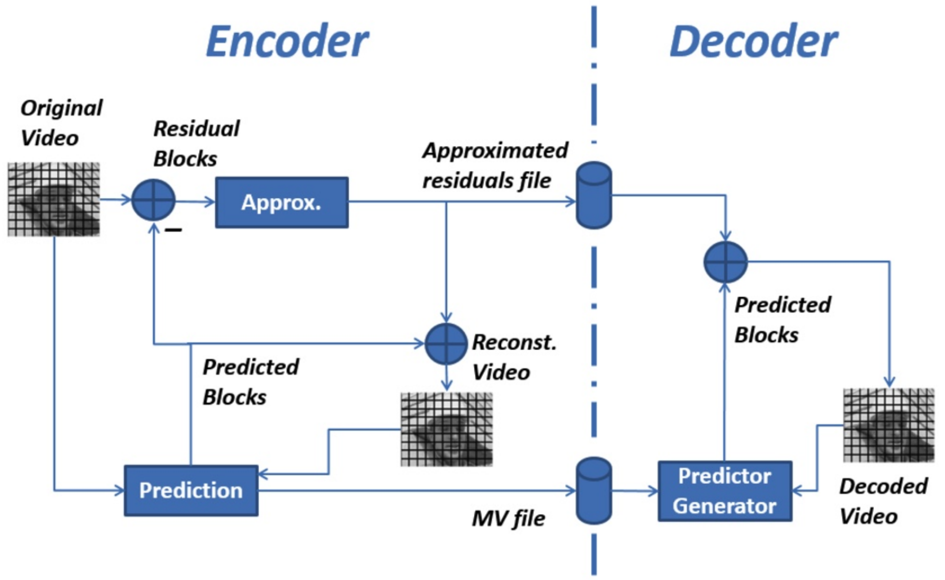
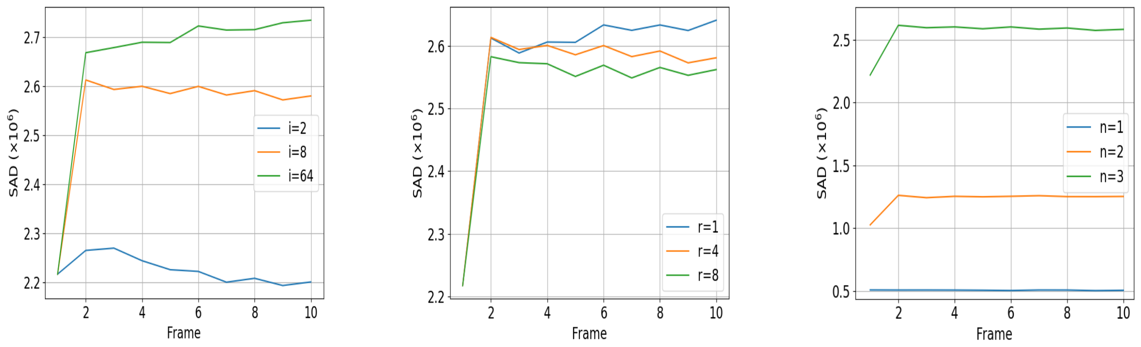


Figure 1: Structure of The Simple Encoder



(a) i sweep with $r = 4$ and $n = 3$. (b) r sweep with $i = 8$ and $n = 3$. (c) n sweep with $i = 8$ and $r = 4$

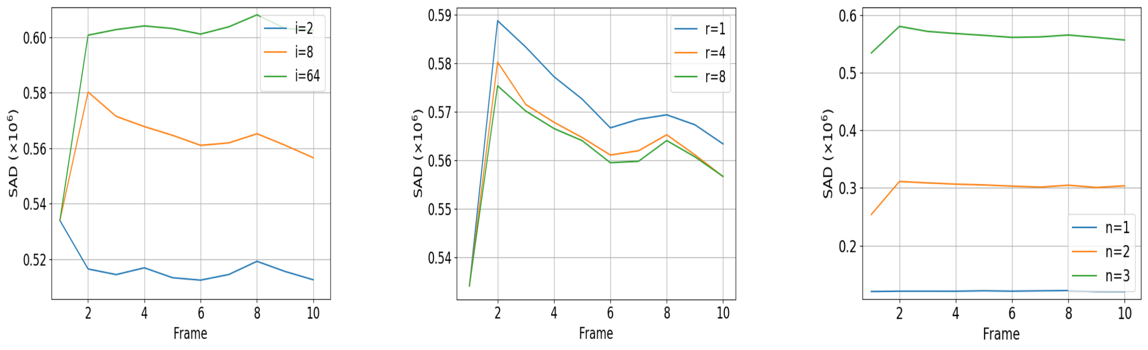
Figure 2: Per-frame SAD graphs for the Foreman video.

2.1 Experimental Results

Fig. 2 shows per-frame Sum of Absolute Differences (SAD) graphs for different encoder settings for the benchmark 352×288 Foreman CIF video. In Fig. 2a $r = 4$ and $n = 3$ were fixed and a sweep on i was performed in $[2, 8, 64]$; in Fig. 2b $i = 8$ and $n = 3$ were fixed and a sweep on r was performed in $[1, 4, 8]$; in Fig. 2c $i = 8$ and $r = 4$ were fixed and a sweep on n was performed in $[1, 2, 3]$. Fig 3 shows a similar analysis for the 176×144 Hall Monitor video.

Analyzing first the i sweep, the experimental result corroborates the theoretical assumptions of Section 2: lower i values lead to lower SAD. This is intuitive, since smaller individual blocks imply more blocks in total to cover a frame, which in turn leads to more motion vectors and greater chances of finding strongly correlated blocks for each one of these smaller blocks.

Unfortunately, smaller block sizes are not the panacea for all encoding solutions. As mentioned above, more blocks imply more motion vectors and overall more calculations, which can be prohibitive especially



(a) i sweep with $r = 4$ and $n = 3$. (b) r sweep with $i = 8$ and $n = 3$. (c) n sweep with $i = 8$ and $r = 4$

Figure 3: Per-frame SAD graphs for the Hall Monitor video.

for weaker hardware (e.g.: embedded systems), battery powered devices (e.g.: phones) or hardware where CPU die temperature can or should not raise too much (e.g.: handheld devices).

Secondly, let us analyze the effect of the block search range r. Again, the theoretical assumptions are endorsed by the experimental data: larger r leads to better results. This happens because the search space for correlated blocks is larger, thus improving the chances of smaller residues.

Regrettably, for the same reasons why one should not simply make i as small as possible, one should not make r as large as possible. That is, the computational, energy and thermal costs might be prohibitive for some applications.

Thirdly, focusing on n, again theory matches practice: lower approximation leads to better results. Although approximation does not impact too much on the execution time of the encoding/decoding process, it can significantly change the quality and final file sizes. This parameter should be chosen based on a device's storage capacity and the demanded output quality.

Finally, it is worth making a joint analysis of all the parameters. Clearly n has the most significant impact on quality. The parameter i comes second, and as shown by Fig. 2a and Fig. 2c, when sufficiently small it can even pull down the higher SAD caused by higher quantization. For this analysis, notice that when $n = 3$ the baseline on Fig. 2c is around $\text{SAD} = 2.5 \times 10^6$, but with $i = 2$ the SAD for the same n was reduced to approximately $\text{SAD} = 2.2 \times 10^6$.

The changes imposed by r are not as dramatic in the chosen range, but are still relevant. As a generalization, parameters can be understood individually, but the best combination can only be determined empirically and for a certain application domain. As discussed above, i, r and n are not strictly decoupled.

The plots in Fig. 3 are also consistent with the above analysis. This may be an indication that the arguments presented above are general enough for different video sequences. It is also worth noticing that the maximum SAD is considerably lower for this video sequence when compared to the one analysed in Fig. 2, which is explained by the fact that the Hall sequence has almost no moving elements in the frame, making estimates and residues easier to deal with. Also, because this sequence's resolution is lower, there are fewer samples to accumulate on the SAD.

Another way of interpreting how i and r influence the accumulated absolute residual magnitudes per frame and also the total time to encode/decode videos is in tabular form, as in Table 1.

Table 1: Influence of i and r on the accumulated absolute residual magnitude (RM) per frame and encoding/decoding times

i = 2 — r = 2										
Frame RM	5492600	143920	184432	159824	142480	123080	90384	94536	74824	73680
Encoder time: 49.33s — Decoder time: 2.63s										
i = 8 — r = 2										
Frame RM	5492600	241952	292872	255896	250648	221992	171344	194608	160352	168304
Encoder time: 9.44s — Decoder time: 1.52s										
i = 64 — r = 2										
Frame RM	5492600	393760	469784	413336	418048	394680	348688	361280	338712	352592
Encoder time: 8.03s — Decoder time: 1.69s										
i = 2 — r = 4										
Frame RM	5492600	71720	89136	77616	74352	64512	48904	54600	47064	49928
Encoder time: 133.74s — Decoder time: 2.69s										
i = 8 — r = 4										
Frame RM	5492600	180216	208184	180416	187824	167696	127248	154640	130000	144536
Encoder time: 15.17s — Decoder time: 1.60s										
i = 64 — r = 4										
Frame RM	5492600	354608	423112	366992	381424	371680	326656	344880	328656	345072
Encoder time: 8.44s — Decoder time: 1.73s										
i = 2 — r = 8										
Frame RM	5492600	40688	43856	40904	44120	41064	32736	39744	36984	38208
Encoder time: 429.69s — Decoder time: 2.72s										
i = 8 — r = 8										
Frame RM	5492600	132848	150136	131784	146904	138008	107672	142312	122344	136672
Encoder time: 34.03s — Decoder time: 1.67s										
i = 64 — r = 8										
Frame RM	5492600	332496	391408	347144	366416	359816	322920	344912	328784	345392
Encoder time: 8.91s — Decoder time: 1.72s										

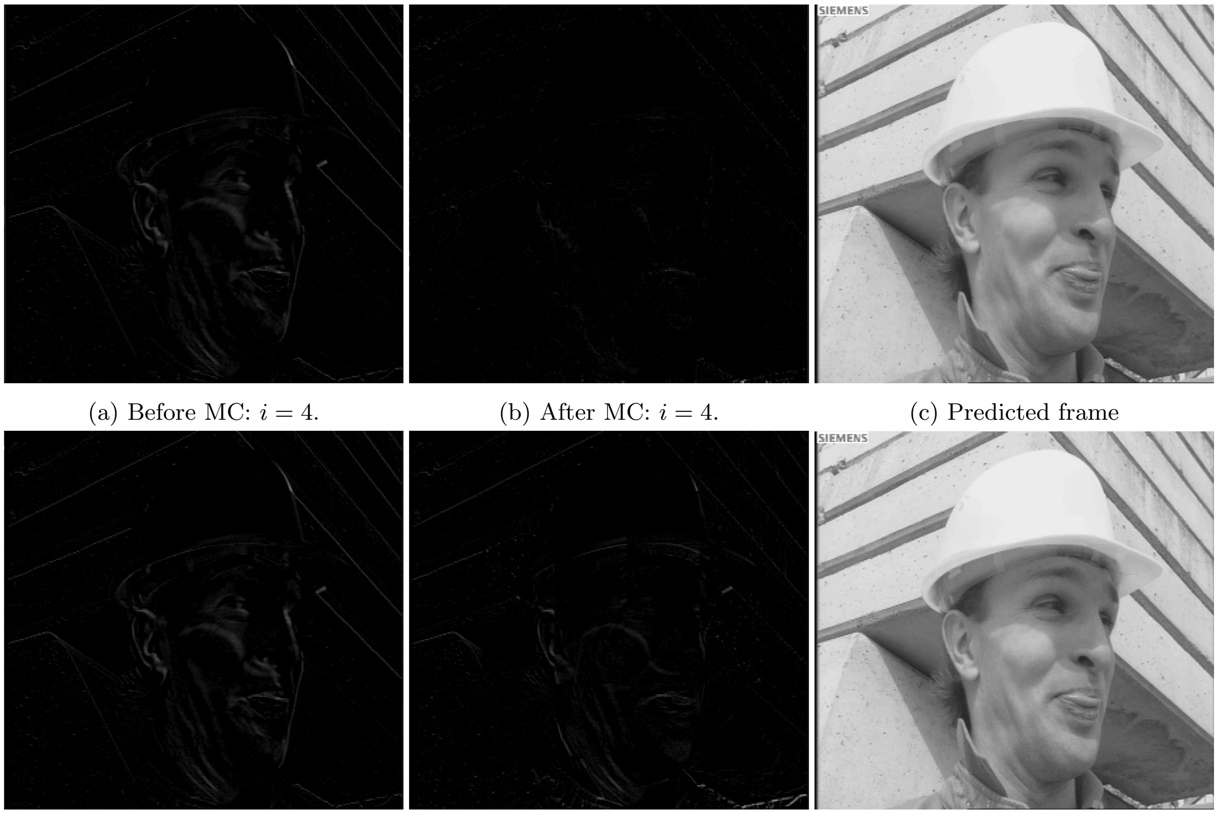
Again it is noticeable that as i grows so does the absolute residue per frame. For a fixed i the larger the r the smaller the residue. The execution time increases inversely proportional to i and directly proportional to r. For the specific combinations experimented with this can be noticed particularly on the combination $i = 2$ and $r = 8$, which takes 429.69s to encode.

In terms of decoding time, it behaves similarly to the encoding. However, the magnitudes of the times are considerably smaller. This is expected since decoding is significantly less burdensome in terms of computational complexity than the reverse process.

Not surprisingly the first frame has the largest accumulated residue. This happens because it uses as a reference for reconstruction a purely gray frame that is not necessarily correlated to the content on the video.

Fig. 4 show the residual frame before and after motion compensation as well as the predicted frame for $r = 4$, $n = 2$, and frame number 8 for $i = 4$ and $i = 64$. As expected, the residual frame after motion compensation is smaller (more dark pixels) than the counterpart before motion compensation.

As expected, when comparing Fig. 4b and Fig. 4e, the residuals for the smaller i value are less noticeable



(a) Before MC: $i = 4$. (b) After MC: $i = 4$. (c) Predicted frame

(d) Before MC: $i = 64$. (e) After MC: $i = 64$. (f) Predicted frame.

Figure 4: Residuals before and after motion compensation (MC) and predicted frame. In all cases $r = 4$, $n = 2$, frame : 8.

(for $i = 4$ the residual is essentially a full black frame, whereas for $i = 64$ the outline of the face can still be seen). This happens because small blocks allow a fine-grained reconstruction of the frames, as discussed above.

3 The Elaborate Encoder

Fig. 5 shows the structure of the Elaborate Encoder. This design can be viewed as a more realistic[2] encoder built on top of the Simple Encoder. In addition to basic approximation and prediction, the Elaborate Encoder adds quantization in a transformed domain, differential encoding, and entropy compression in order to achieve significantly smaller output file sizes.

The Quantization Parameter (QP) determines the crudeness of the quantization, with QP = 0 being the most refined level and the one that should theoretically lead to the highest video quality. Conversely, with each increment in QP the output video quality should progressively degrade.

Another adjustable parameter is the I_period, which sets the number of p frames between 2 successive I frames (plus one I frame). In this context, I frames are those in which intra-prediction is performed, that is, no information from previous frames are used in their encoding. The p frames are just the opposite; they are encoded based on differential information from previous frames. The advantage of having I

[2]From the point of view of commercially available encoder/decoder technologies

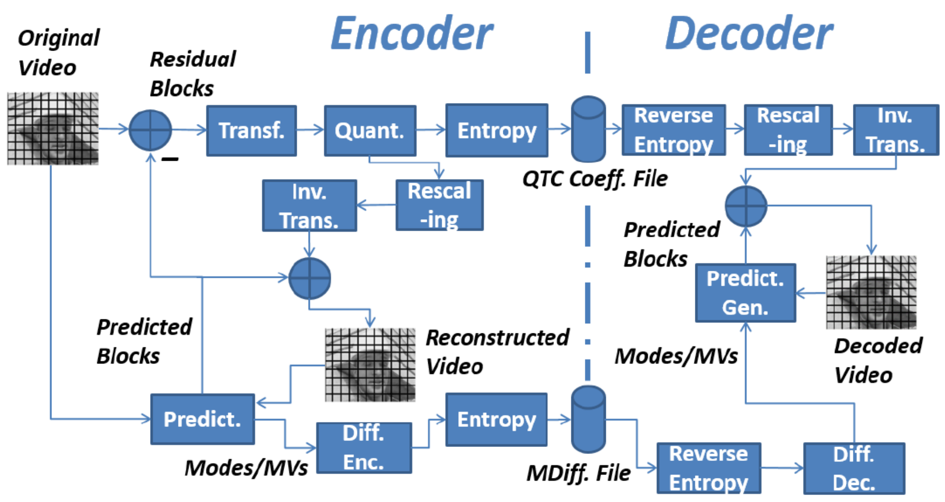


Figure 5: Structure of The Elaborate Encoder

frames in a video is that they provide independent synchronization points and allow recovery of the prediction stream when data loss occurs.

Entropy compression is performed twice in this encoding process. In one instance the goal is to reduce the number of bits required to represent the quantized transformation coefficients; in the other, the target is compressing the prediction information.

In the first case compression is mainly made possible because redundancy in the residual data is augmented by pre-processing strategies. The stages are as follows: discrete cosine transform to find spatially concentrated energy packets, followed by quantization to round and maximize reiteration of values. At this point, a lot of redundancy is already explicit in the data, and it can be made even more evident if matricial spatial data is projected into 1D spaces with a scanning technique that orders values from highest to lowest. Repeating values can then be converted into counts and, finally, be encoded into an efficient bitstream that uses fewer bits for more common values.

In the second case, differential encoding is performed on the intra-prediction modes and motion vectors. For the intra-prediction case, this strategy will lead to two possible values meaning “the same as before” or “change”. Because neighboring blocks in an ergodic video should be similar, it means a lot of repetition in these values is expected, which is exactly what entropy compression is specialized in dealing with. The differential encoding of motion vectors follow a similar logic: neighboring blocks should move in similar directions, leading to near zero differences that can be compacted by the downstream entropy block.

3.1 Experimental Results

Fig. 6 and Fig. 7 show rate distortion (R-D) plots for block size $i = 8$, $i = 16$ and search range $r = 2$. The rates in the x-axis were reached by scanning the quantization parameter QP in [0..9] with unitary increments when $i = 8$ and in [1..10] when $i = 16$. The multiple curves in the plot were achieved by changing the I_period. The results are discrete points that were connected with the lines presented in the charts. It is important to highlight that these results were obtained from real encoder output, with Golomb coding enabled, as in the source-code that is packaged with this report.

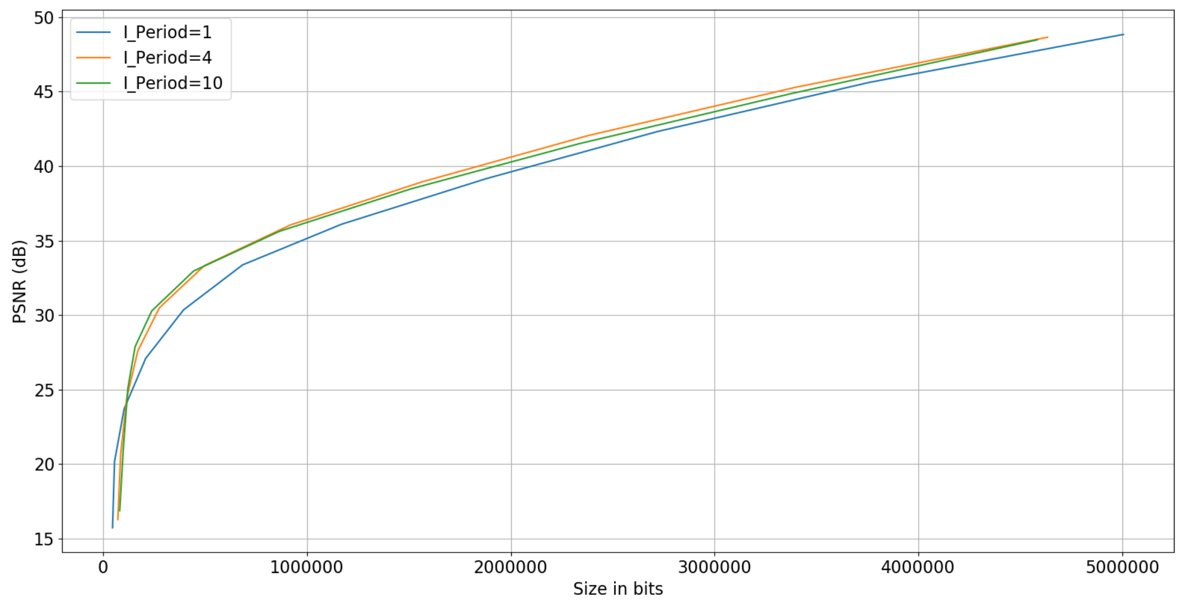


Figure 6: Sequence Size vs. PSNR with $i = 8$.

With the aid of these plots we can conjecture about the relevance of QP and I_period on this specific video sequence. By sectioning the plot horizontally (i.e.: fixing Y) it becomes clear that to reach a certain quality level with only I frames more bits are required in general. This, however, does not mean that fewer I frames necessarily lead to more bits in the stream, since I_period = 4 outperforms I_period = 10 in most of the x-axis range in terms of PSNR per bit.

It is noteworthy that I frames do not consume more bitrate across all the QP rate. As this parameter reaches its maximum, the PSNR drops dramatically, and at this point an I frame only sequence may actually demand fewer bits for the same quality level, as can be noticed in the bottom left corner of Fig. 6. Depending on the chosen i this effect might not be as noticeable - similarly to Fig. 7 - however, in this case the performance difference between the I frame only sequence and the ones including p frames became marginal.

The aforementioned insight is technically justified by further interpreting what the I_period signifies. This parameter is the periodicity at which a new frame that is independent from past frames in the stream is generated. If the distance between I frames is too long, p frames will start to accumulate a lot of error in their predictions. When an I frame comes it resets the error, and a new cycle of accumulations begin. Because larger errors require more bits to be encoded, that is why a balanced I_period needs to be empirically determined.

In terms of compression ratio versus PSNR, setting $i = 8$, $r = 2$, QP = 3 and I_period = 4 the achieved compression ratio was 5.17 for the first 10 frames of the benchmark Foreman video, with an average $\text{PSNR}_{\text{AVG}} = 33.02\text{dB}$

Fig. 8 presents plots for the execution time to encode and decode the first 10 frames of the Foreman reference video with diverse settings. Clearly smaller i leads to higher encoding times, and for a fixed i the larger the I_Period the slower the process. When decoding, smaller i also implies longer run times, but the I_Period has proportionally much less influence in the decoding times, with curves almost overlapping.

It is easy to understand why smaller i leads to longer encoding/decoding times, after all, the smaller this parameter, the more blocks are there to process, the more motion vectors are there to work with and the more information there is to process overall. When decoding, the same applies.

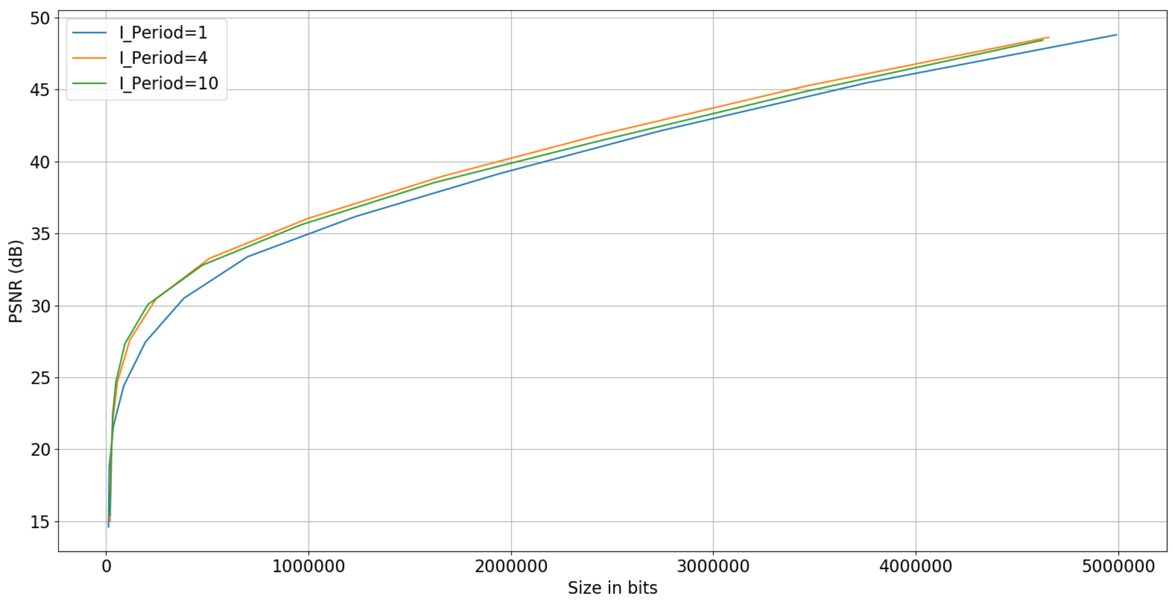


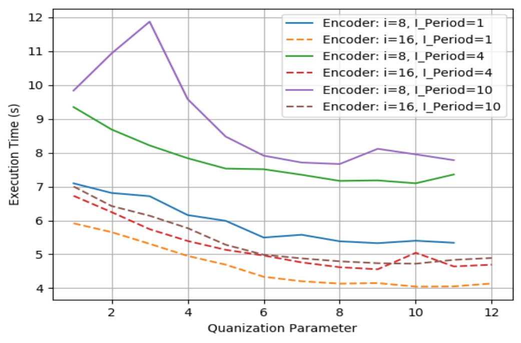
Figure 7: Sequence Size vs. PSNR with $i = 16$.

Regarding the influence of the I_Period, the lower it is, the faster the encoding runs. In other words, the more I frames there are, the faster the process. This is also intuitive: with fewer p-frames there are fewer iterations searching for potential correlated matches in the $r \times r$ space, which is a very time consuming task.

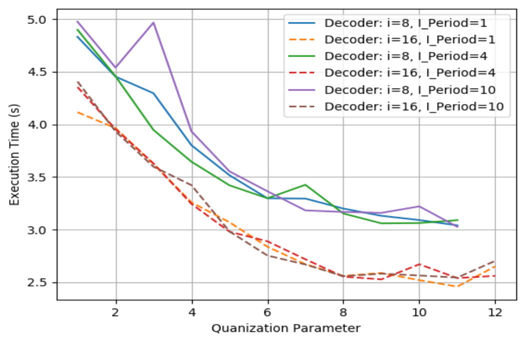
Fig. 9 shows plots for frame index vs. bit-count for $i = 8$ and QP $= 3$ / $i = 16$ and QP $= 4$ with varying I_Period in $[1, 4, 10]$.

The plots confirm the expected theoretical results:

1. For larger QP fewer bits are demanded per frame;
2. For I_Period $= 1$: no p-frames, so each frame requires nearly the same number of bits;
3. For I_Period $= 4$: the graph peaks at each I-frame (i.e frame $=$ 1,5,9) and reaches valleys on

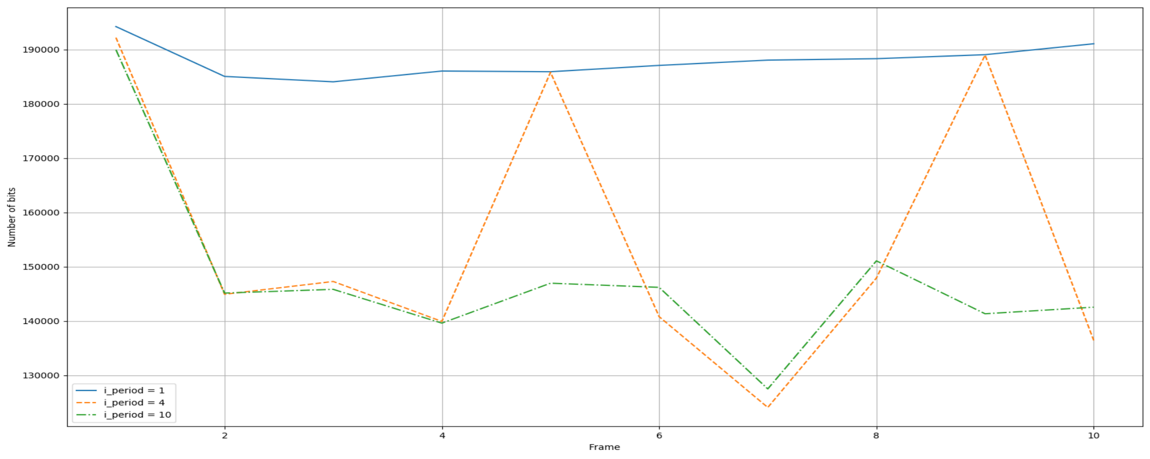


(a) Encoder.

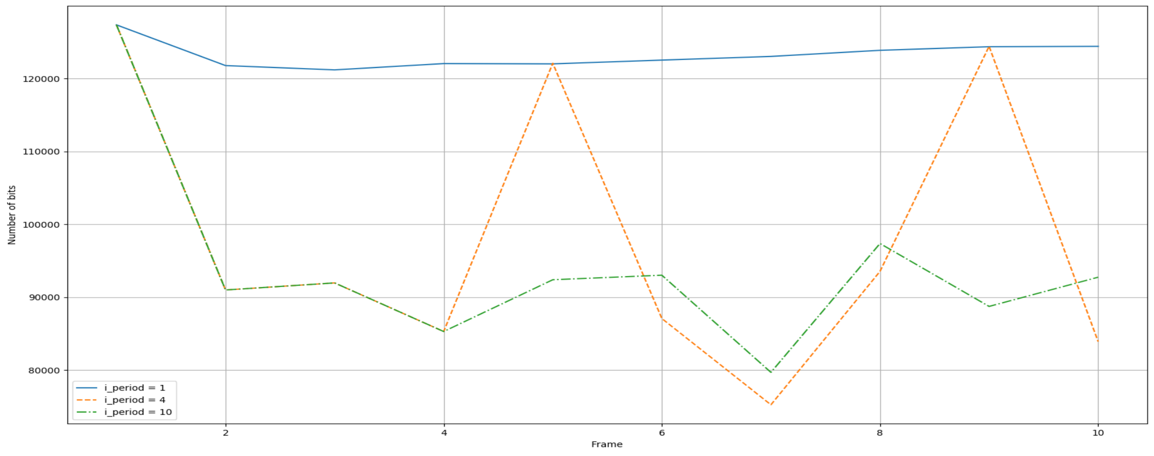


(b) Decoder.

Figure 8: Execution times with varying parameters.



(a) $i = 8$ and QP $= 3$.



(b) $i = 16$ and QP $= 4$.

Figure 9: Frame Index vs. Bit-Count

p-frames. These valleys tend to be more prominent immediately following an I-frame, when the accumulated prediction error is minimal.

4. For I-Period $= 10$: there is only 1 I-frame in the observation range, which is the first frame. There, the graph attains its highest bit-count, followed by dips induced by the p-frames lower bit demands.

5. Comparing the minimum valley for I_Period $= 4$ and I_Period $= 10$ (on frame 7), it is noticeable that the latter has a lower bit count. This happens for the same reason as pointed out during the analysis of Fig. 6 and Fig. 7.

4 Specific Challenges and Simplifications

The use of the Python programming language allowed many simplifications, especially in combination with the Numpy and Scipy packages. The former allowed operations in multidimensional data to be carried with one-line commands that would be burdensome in other programming languages. The latter provided the Discrete Cosine Transform as a packaged solution that simplified the development of the Elaborate Encoder. Also, Python's native syntax to slice multidimensional data arrays was extremely handy when sectioning frames.

On the other hand, because Python is not inherently explicit about data types, a lot of extra effort had to be made to keep track of the byte size of our variables. Saving bitstreams with Python is also less intuitive than with languages like C/C++.

Regarding the engineering aspects of the encoding/decoding process, some areas were particularly more complicated to comprehend:

1. Padding was necessary for certain values of 'i'. This made it intricate to keep track of padded blocks and decide whether to include them or skip them while performing operations like 'Predictions' and 'Entropy'.
2. When expanding the Simple Encoder into the Elaborate Encoder it was not immediately clear that the initial gray frame was no longer needed, which caused confusion about which should be the first 'I' frame.
3. Also, it was not immediately clear when to use blocks of the original video or blocks of the reconstructed frame. This confusion was quickly mitigated as more familiarity with the philosophy of the encoding/decoding process was built.
4. Another convoluted aspect was implementing the Exponential Golomb encoding and building bitstreams. It was not straightforward to understand how to represent data with size inferior to 1 byte without appending zeros in the most significative bits to "complete" the byte.
5. Figuring out the meta data needed to be passed to the decoder was not straightforward.

5 Conclusion and Future Work

In this report two video encoders/decoders were implemented, one with a simple structure where only prediction and basic approximation of residual values was performed, and the other followed a more robust approach, fully described in Section 3. Based on all the information presented on this report and also on how empirical results match theory, it is safe to assume the encoders/decoders are working as they should.

Experimental results were made available for the reader and analyzed in depth. Some of the most important conclusions are summarized below:

1. Smaller Block Size (i) and Quantization Parameter (QP) generally lead to better quality output video. In the case of the Search Range (r), larger values maximize the odds of finding correlated blocks, leading to smaller errors and therefore better video quality.
2. From Fig. 9 it can be derived that bit-rate is directly related to the relative count of I-frames versus p-frame. I-frames consume larger chunks as compared to p-frames.

Some suggestions for future work include: performing run-length encoding on the motion vectors, using massively parallel design (such as GPU aided processing) to speed up the encoder/decoder, take measures to minimize memory usage during run-time, implement b-frames as well and add smartness to the run length encoding such that zeros are only encoded in runs when this would lead to fewer bits.