ECE1783 Assignment 1 Report

Group:

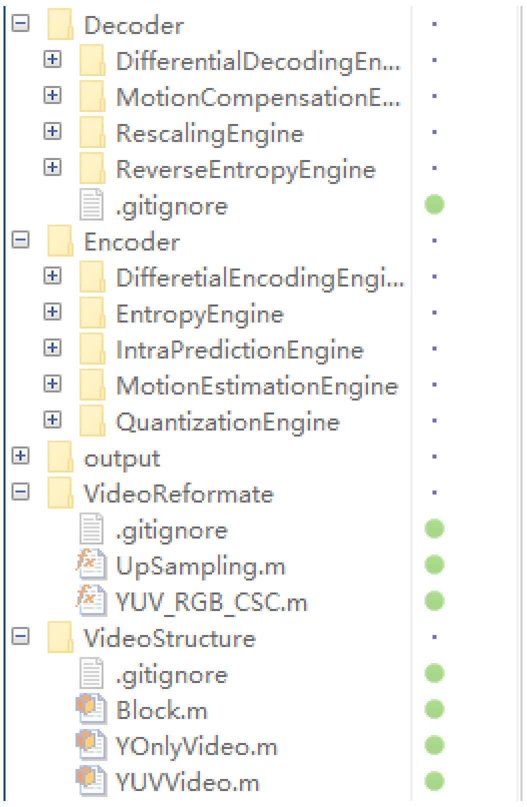
Ziwen Wang 1007350242
Pranav Venkatesan 1006593483
Javert Zhang 1002689003

Introduction

Nowadays, video encoding is extremely important because it allows us to more easily transmit video content over the internet. In video streaming, encoding is crucial because the compressing of the raw video reduces the bandwidth making it easier to transmit, while still maintaining a good quality of experience for end viewers. In this assignment, our group build a small encoder engine that encodes the Y component of input YUV video and send to decoder side. In addition, we also build a decoder to decode the encoded data and transform the data back to video.

Approach

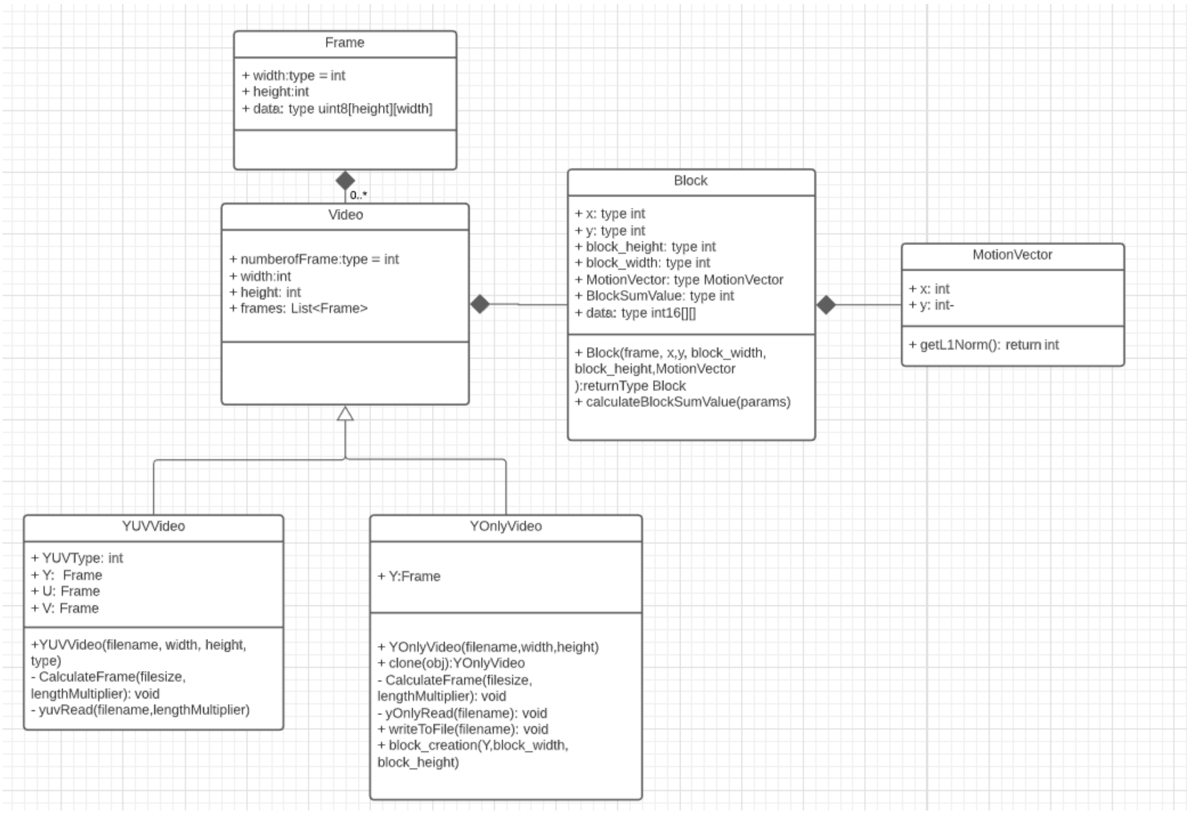
Our engine is written in Matlab. Since the whole encoder engine is relative big project, we break up the engine to different small engines. Here is example of showing our code structure in terms of directory view:



```
⊟ Decoder
    ⊞ DifferentialDecodingEn...
    ⊞ MotionCompensationE...
    ⊞ RescalingEngine
    ⊞ ReverseEntropyEngine
      .gitignore
⊟ Encoder
    ⊞ DifferetialEncodingEngi...
    ⊞ EntropyEngine
    ⊞ IntraPredictionEngine
    ⊞ MotionEstimationEngine
    ⊞ QuantizationEngine
⊞ output
⊟ VideoReformate
      .gitignore
      UpSampling.m
      YUV_RGB_CSC.m
⊟ VideoStructure
      .gitignore
      Block.m
      YOnlyVideo.m
      YUVVideo.m
```

Data Structure

All the pixel data are stored as structure of matrix. Besides that, we also wrote some customized data structure to simplify process during the encoding. We have plot a rough UML design for our data structure:



Frame
+ width:type = int
+ height:int
+ data: type uint8[height][width]
0..*
Video
+ numberofFrame:type = int
+ width:int
+ height: int
+ frames: List<Frame>
Block
+ x: type int
+ y: type int
+ block_height: type int
+ block_width: type int
+ MotionVector: type MotionVector
+ BlockSumValue: type int
+ data: type int16[][]
+ Block(frame, x,y, block_width, block_height,MotionVector):returnType Block
+ calculateBlockSumValue(params)
MotionVector
+ x: int
+ y: int-
+ getL1Norm(): return int
YUVVideo
+ YUVType: int
+ Y: Frame
+ U: Frame
+ V: Frame
+YUVVideo(filename, width, height, type)
- CalculateFrame(filesize, lengthMultiplier): void
- yuvRead(filename,lengthMultiplier)
YOnlyVideo
+ Y:Frame
+ YOnlyVideo(filename,width,height)
+ clone(obj):YOnlyVideo
- CalculateFrame(filesize, lengthMultiplier): void
- yOnlyRead(filename): void
+ writeToFile(filename): void
+ block_creation(Y,block_width, block_height)

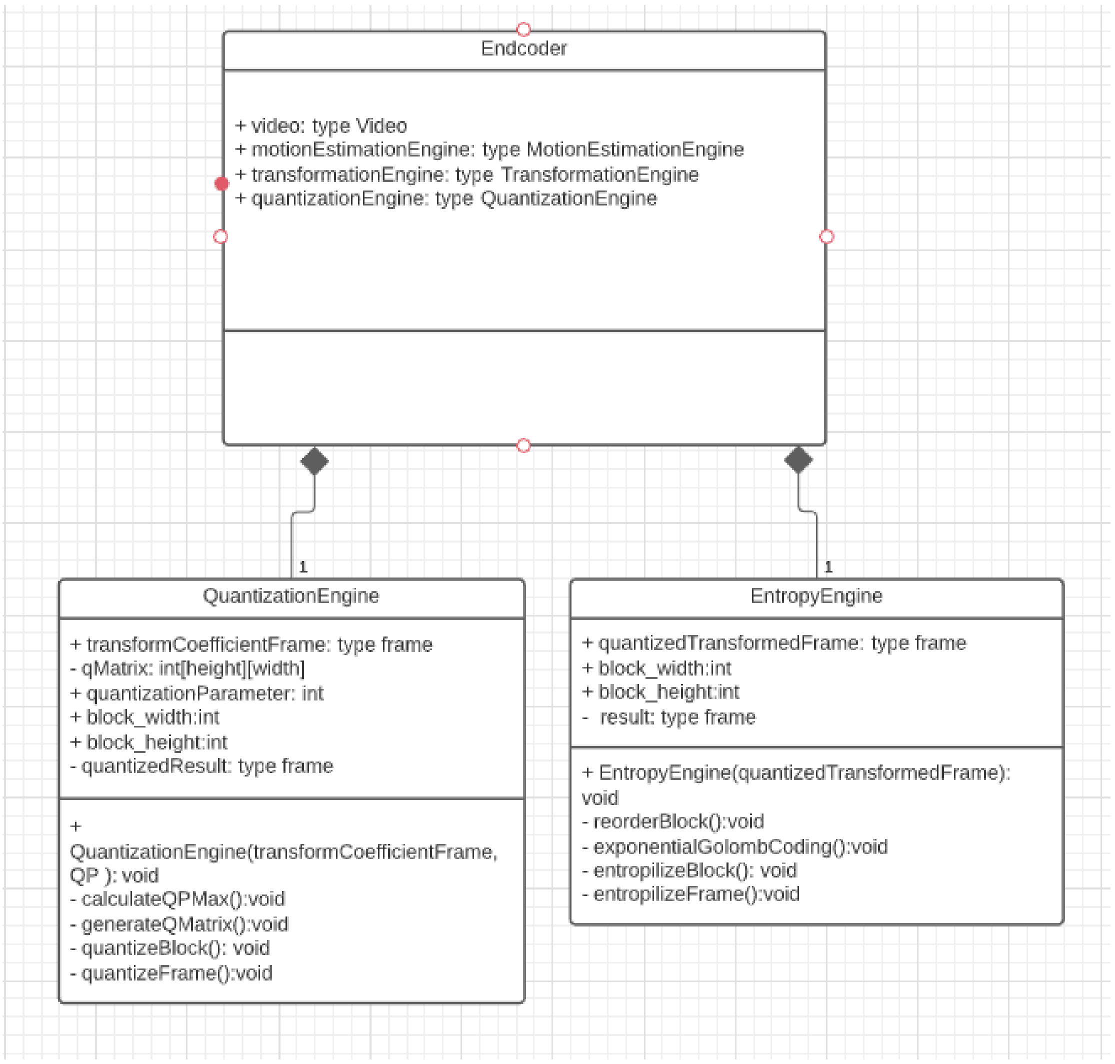
Different class will contain different attributes and method to handle the current class related operations. For example, we have data structure called Block, it contains properties as shown below:

```
classdef YUVVideo
    properties (GetAccess='public', SetAccess='public')
        width;
        height;
        YUVType;
        numberOfFrames;
        Y;
        U;
        V;
    end
```

This class also contains different methods to handle YUV video such as read from file, write to file, calculate number of frame according to input file size, video width and video height etc.

Encoder/Decoder Engine

As mentioned before, our encoder engine has been break down to small parts and each parts handling different operations during the encoder process. The following graph shows a simple example of some components in encoder in UML diagram:

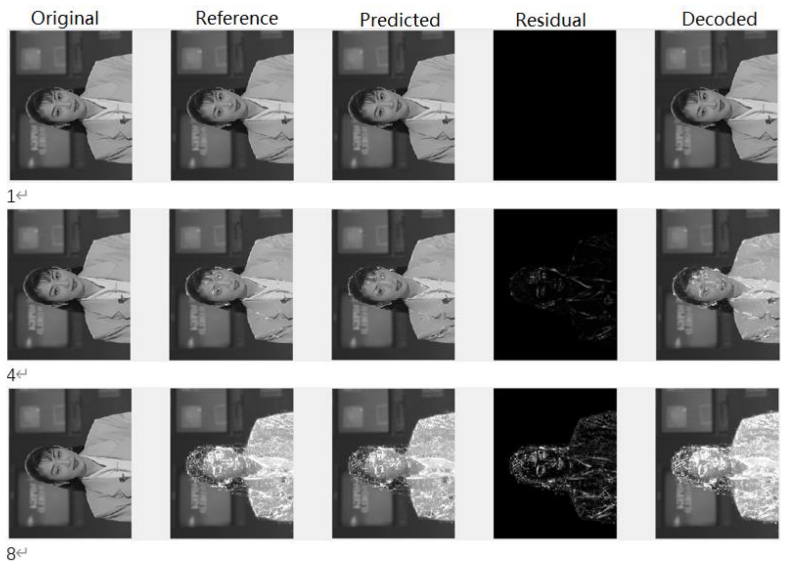

Endcoder
+ video: type Video
+ motionEstimationEngine: type MotionEstimationEngine
+ transformationEngine: type TransformationEngine
+ quantizationEngine: type QuantizationEngine
1
1
QuantizationEngine
+ transformCoefficientFrame: type frame
- qMatrix: int[height][width]
+ quantizationParameter: int
+ block_width:int
+ block_height:int
- quantizedResult: type frame
+ QuantizationEngine(transformCoefficientFrame, QP): void
- calculateQPMax():void
- generateQMatrix():void
- quantizeBlock(): void
- quantizeFrame():void
EntropyEngine
+ quantizedTransformedFrame: type frame
+ block_width:int
+ block_height:int
- result: type frame
+ EntropyEngine(quantizedTransformedFrame): void
- reorderBlock():void
- exponentialGolombCoding():void
- entropilizeBlock(): void
- entropilizeFrame():void

As you can observe from the graph, the Encoder engine contains Quantization Engine and Entropy engine. The quantization engine will take input block/frame from Encoder engine and process it with quantization method. When it completes the task, it will then return the result back to Encoder Engine and Encoder Engine then will pass it to any Engine that requires the quantized result.

All of parameters can be adjust in source code called main3.m and main4.m.

Observation / Challenges / Discussion

1. When we were designing the simple encoder in Part 3, we encountered some discontinuities in the residual frames, and we got decoded frames with some pure white point (255). The problem occurred since we were trying to add two frames of type uint8, where uint8 does not reserve negative values of residual frame. The problem resolved by doing calculation in type of int16.



2. In this assignment, we are uniformly truncate the frame to constant size of blocks. This might not be good idea for real world since for the area in the frame where space are "Temporally-Static", we should use bigger block size. And for the area in the frame where space are "Temporally-Dynamic", we should use smaller block size

3. Although we have carefully designed our code structure in terms of UML graphs, due to the time limitation, we don't have enough time to write detailed documentation for every component inside of our encode/decode engine. We should do that in future so the code can be highly re-useable.

4. We have noticed that the running time of decoding engine is taking extremely long time comparing to encoding process. This is because
   (i) the properties of Exponential-Golomb Coding, which is easy to encode since it know how many bits it should take for the input number. But it does not have that information when it decode the bitstream.
   (ii) Same thing apply to inverse RLE, it takes much longer for decoding comparing to encoding.

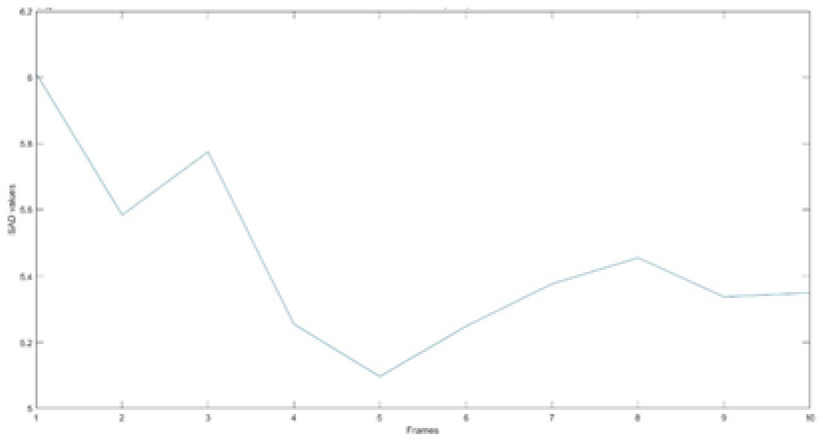
5. For future work, our team think we should either improve the decoding algorithm, relacing decoding method or decode video in parallel so the decode engine will take less time.

Result

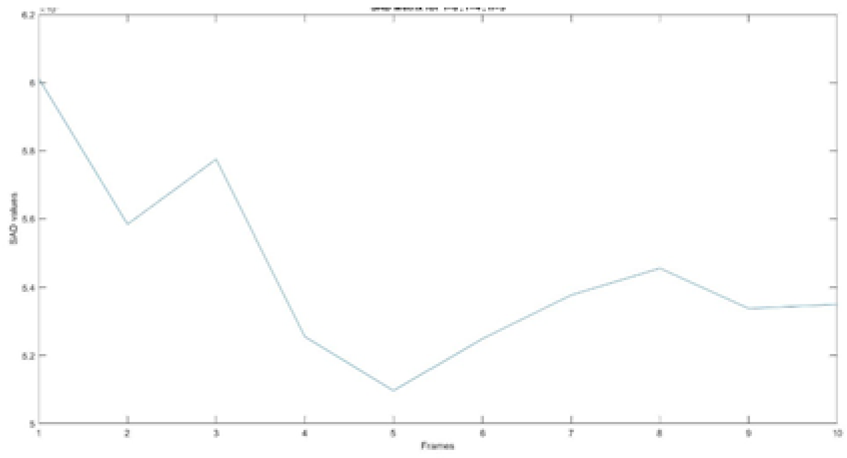
Exercise 3:

1. SAD graphs for r = 4 and n = 3

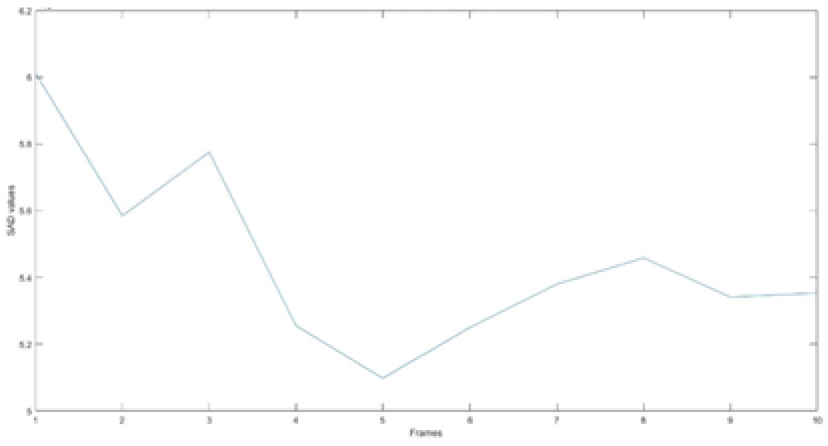
l=2



l=8



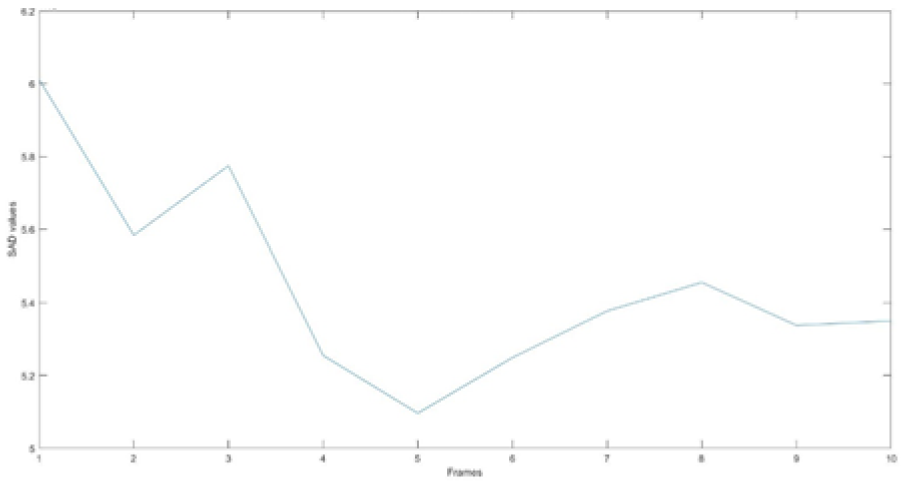
l=64



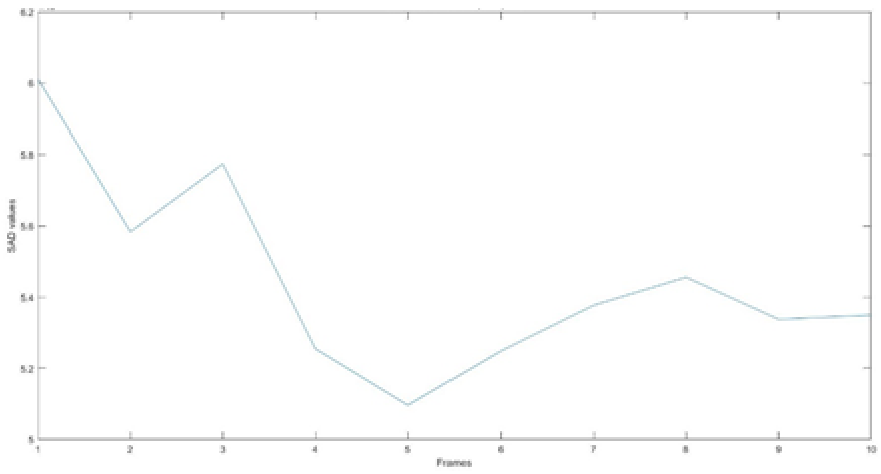
The above images are taken by varying the block size and keeping the search radius and round off values constant. From the above images it can be seen that the SAD values does not change much with the block size.

2. SAD graphs for I = 8 and n = 3

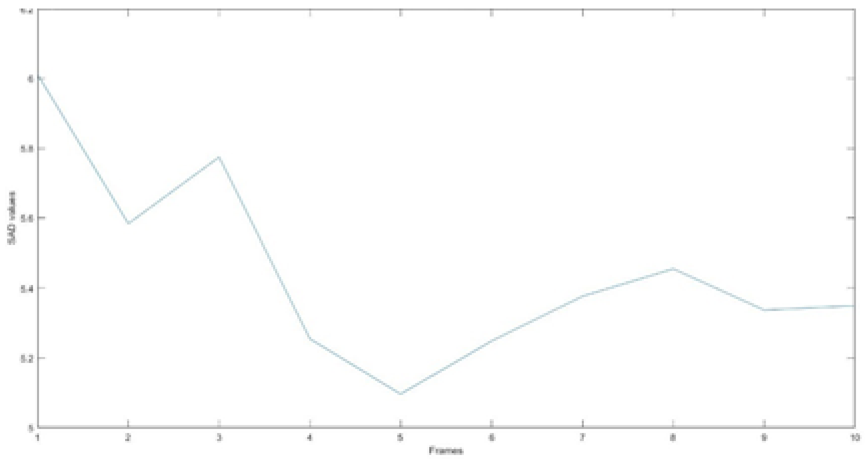
R=1



R=2



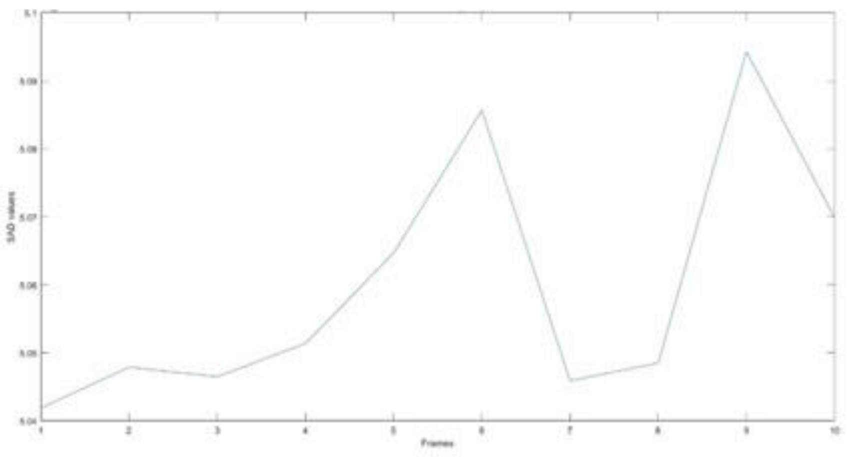
R=4



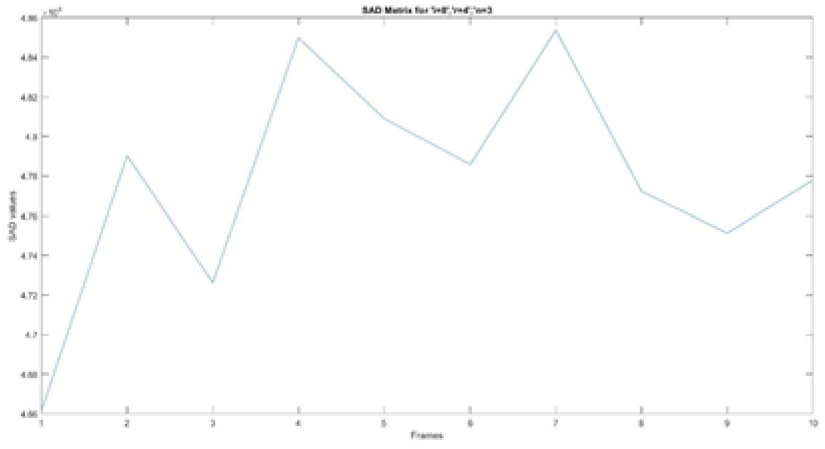
Like the above presented graph, these graphs are drawn with varying the search variable r value by 1,2,4 and keeping the I and n value constant. It is evident that changing the r value alone will not produce much difference between them.

3. SAD graphs for I = 8 and r = 4

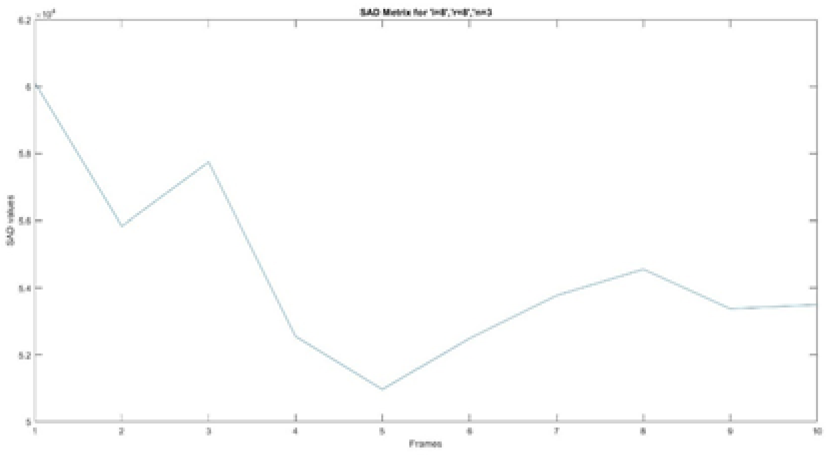
N=1



N=2



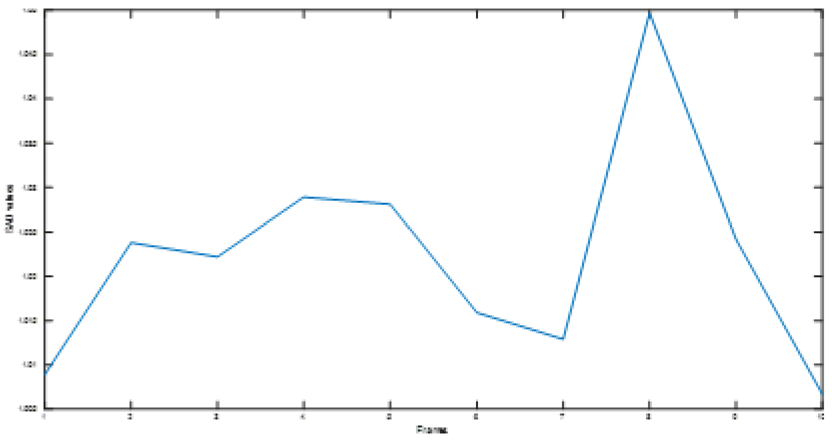
N=3



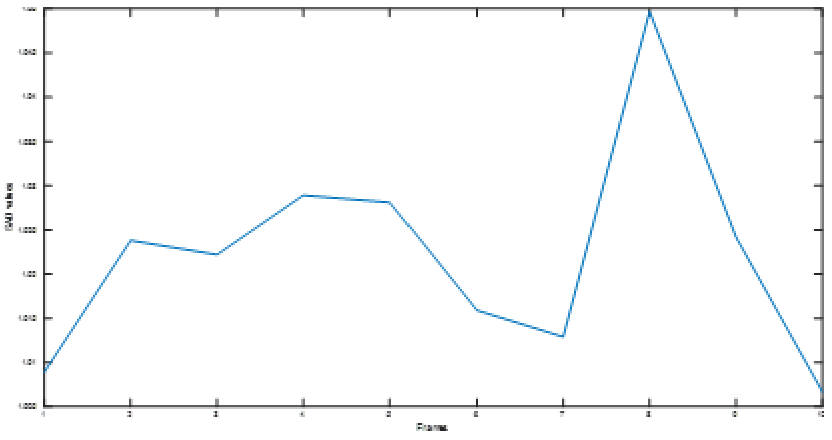
The above graphs show significant variation in the SAD values among the three. The first graph with n=1, have an uneven trend in the SAD values. For n=2, the SAD values always tends to be higher and lastly for n=3, the SAD values tend to be lower when compared to the other two.

4. SAD for r = 4 and n = 2 for video of different dimension

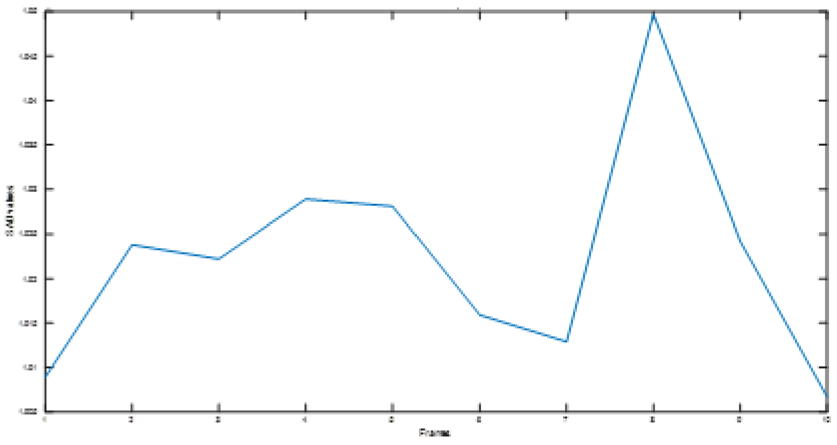
I=64



I=8



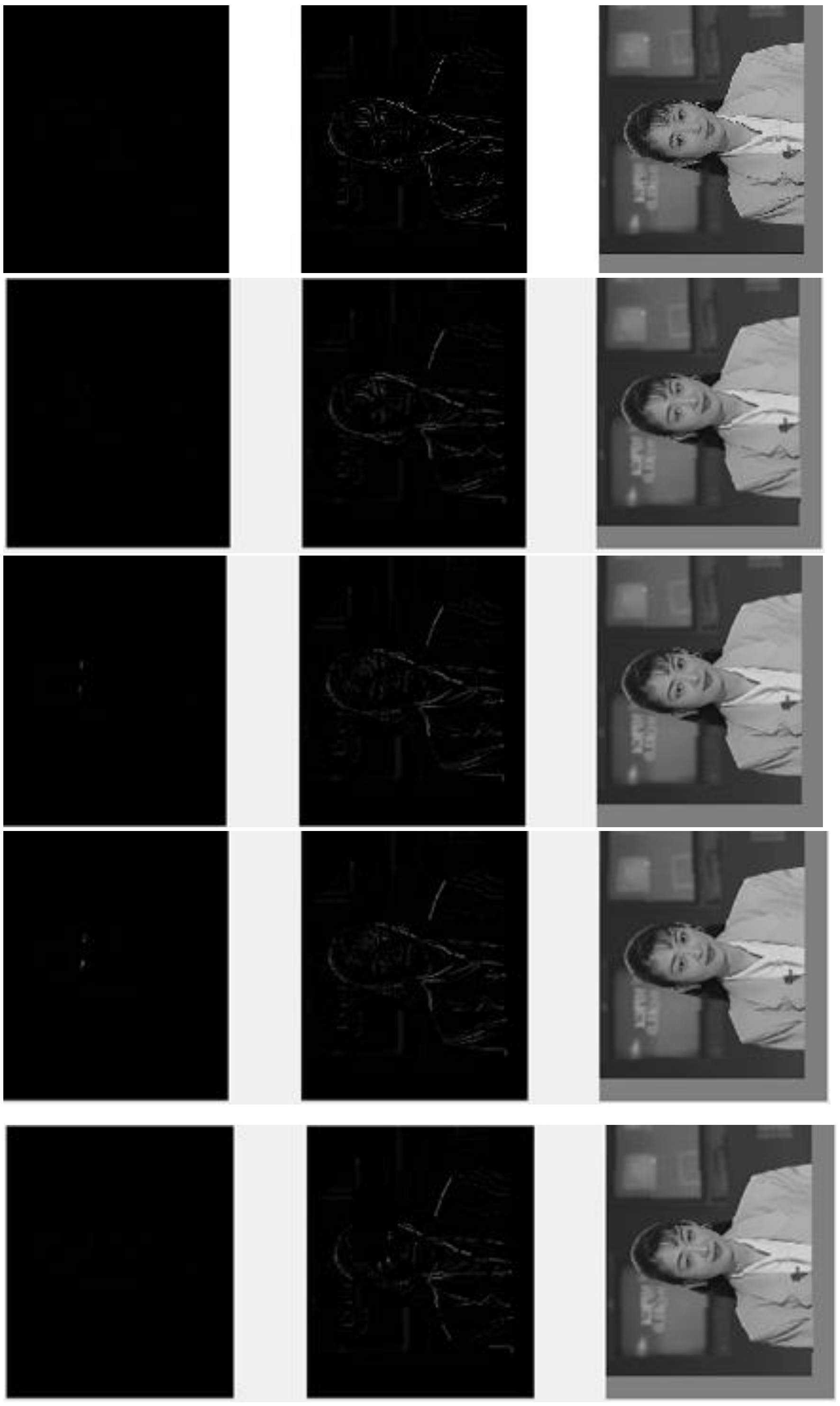
I=2



The above SAD metric graph is made with varying I (64,4,2) with constant r and n for a video of different dimension (176x144) and it show the same behaviour as that is shown for the previous file.

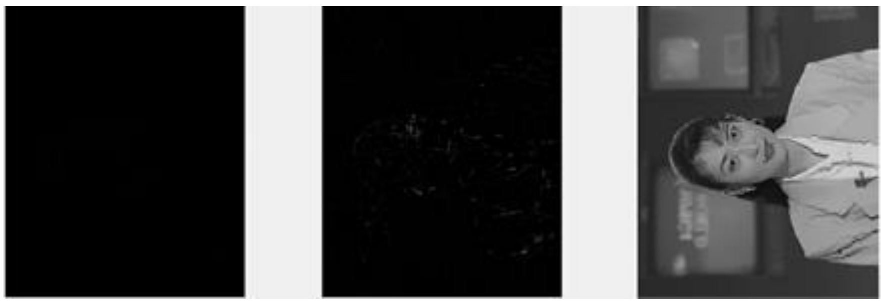
5. Comparison of residual frame before and after motion compensation

(i) I = 64



The above images consist of residue after before motion compensation, after motion compensation and predicted image for the second to sixth frame. These images are taken by considering, i=64, r= 2, n=1.

(ii) I = 8



These images were taken with i=8, r=2, n=1. It can be seen that after-motion compensation the eye part alone if left out and that difference is seen in the predicted image.

6. What is the effect of the i and r parameters on residual magnitude and encoding time? Provide a table that shows how encoding time varies, and another table for residual magnitude.

Table Showing the execution time and residual values for all the combinations of parameters

i	**r**	**n**	**time(s)**	**Residue**
64	1	1	12.153	339157
8	1	1	105.326	225248
2	1	1	511.538	369445
64	2	1	9.382	372148
8	2	1	105.903	339730
2	2	1	1653.149	345508
64	4	1	15.062	405870
8	4	1	576.201	392422
2	4	1	8217.7	382200
64	1	2	7.987	341784
8	1	2	105.32	258720
2	1	2	591.496	278392
64	2	2	8.228	394136
8	2	2	99.825	327908
2	2	2	1508.787	337416
64	4	2	12.935	402708
8	4	2	512.119	385304
2	4	2	8416.587	374088
64	1	3	7.846	374784
8	1	3	40.571	243808
2	1	3	577.712	289760
64	2	3	7.885	403352
8	2	3	97.945	313200
2	2	3	1793.256	367216
64	4	3	18.49	405408
8	4	3	648.282	380992
2	4	3	9262.613	417160

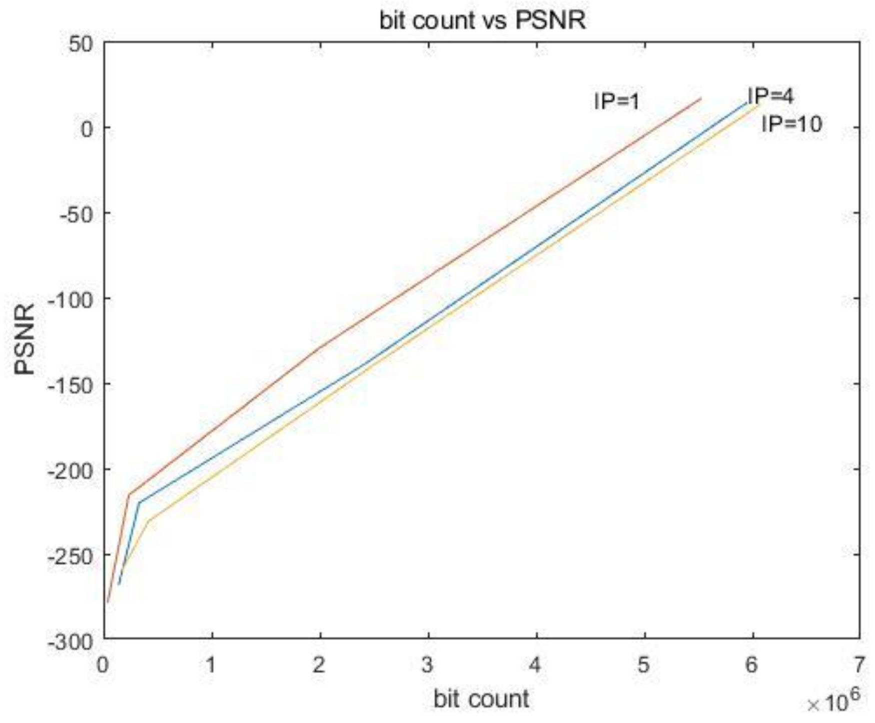
The above tabular column shows the different execution time based on the different i,r,n values. The execution time for all the 27 combinations shave been plotted.
The time taken maintains the same trend by showing increment for n=1,2and 3 for different values of i and r. This effect is same for other combinations for r=1,2,4. But when I value increases the execution time decreases and this trend also follows throughout the various combination. The time is most consumed when **i=2, r=4 and n=3.** The combination that consumes the least amount of time **is i=64, r=1, n=2.**
The residue shows the performance of the engine under different parameters as the input. The above residues are taken for 10 frames. The residue values is the least when i=8. And is larger for both 64 and 2. The residue value increases as r value increases. The same trend is seen when changing the n value as well. This shows that the lowest residue is obtained for the combination i=8, r=1, n=1.

7.What is the impact of each of the parameters (i, r, and n) on the final quality of the image (measured as average SAD between original and reconstructed frames)? Explain the results you are seeing.
The result has been showed and discussed in part 1, 2 and 3 of exercise 3.

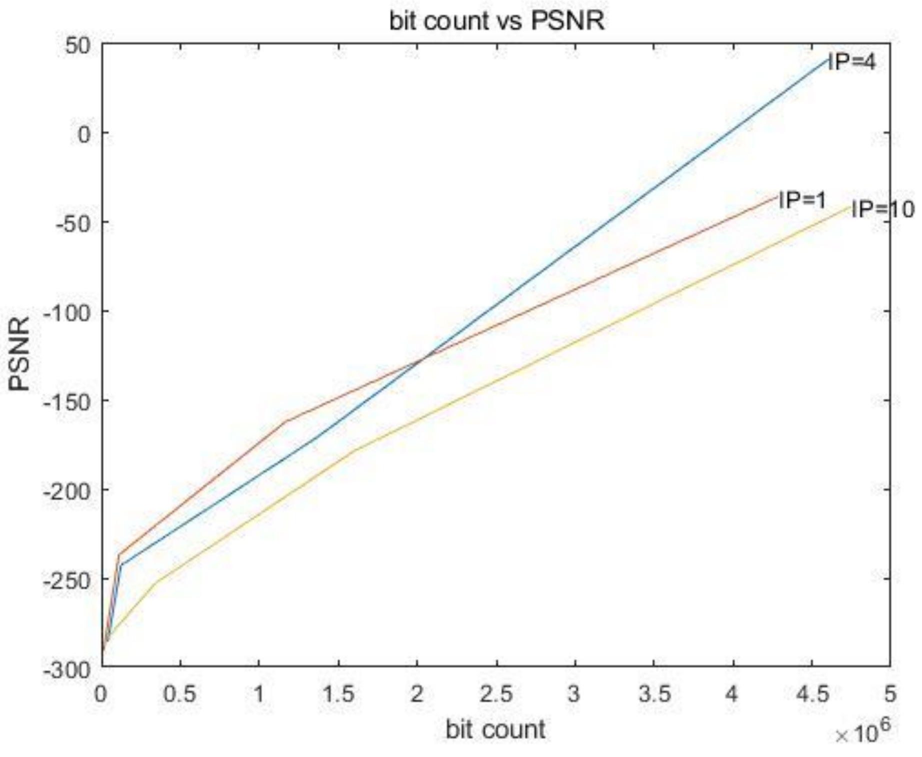
Exercise 4:

1. The following graph shows the R-D plot where the x axis is the total size in bits of a test sequence, and the y axis is the quality/distortion measured as PSNR. All data are obtained by running Foreman video with first 10 frames.

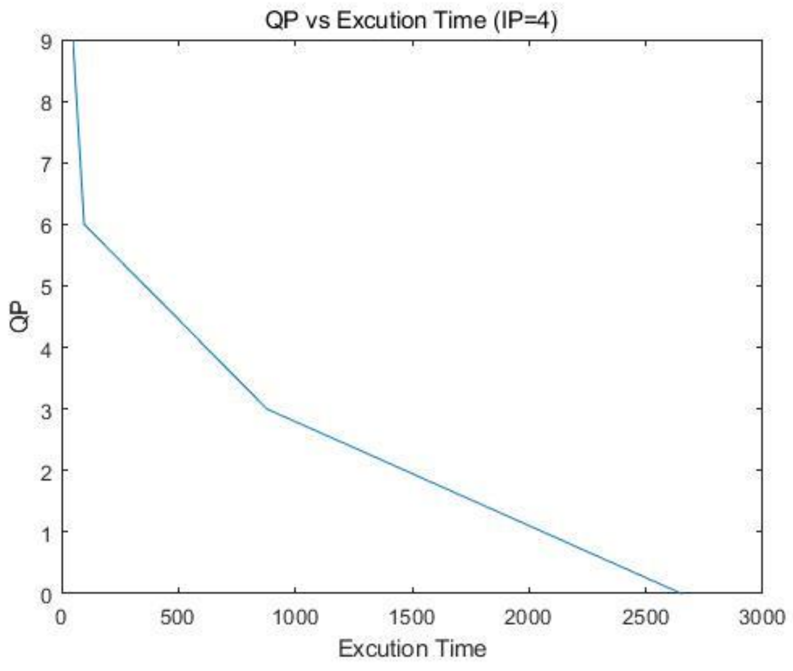
   (i) I = 8



   (i) I = 16



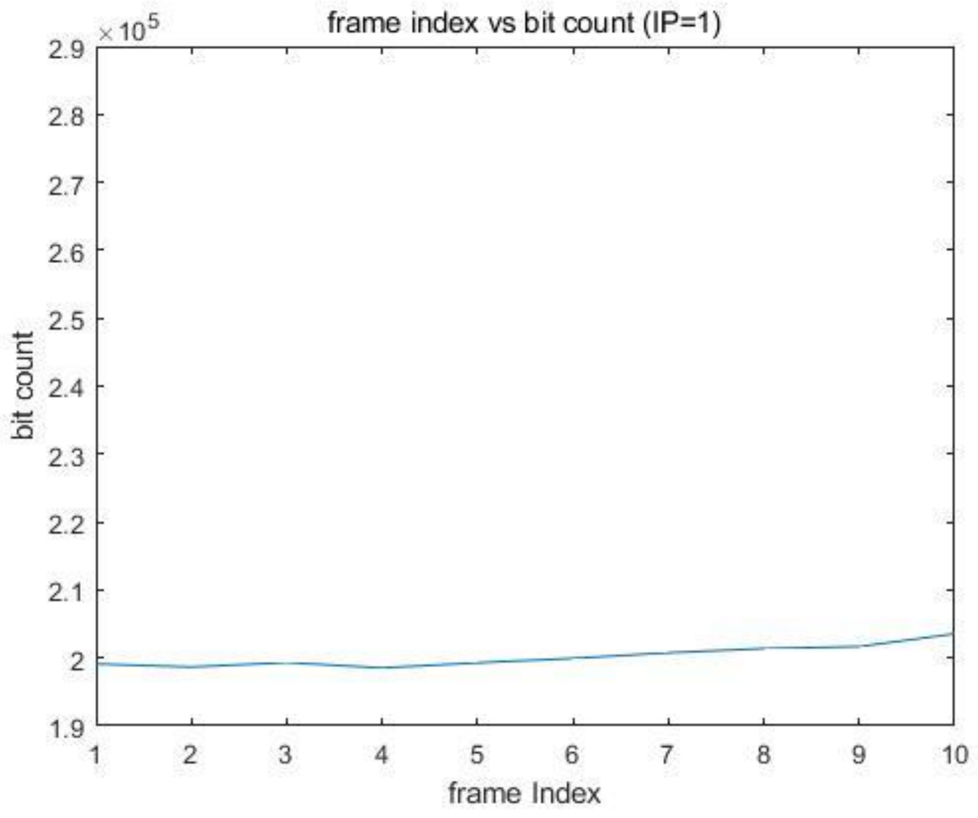
The following graph shows the relationship between execution time and QP value:



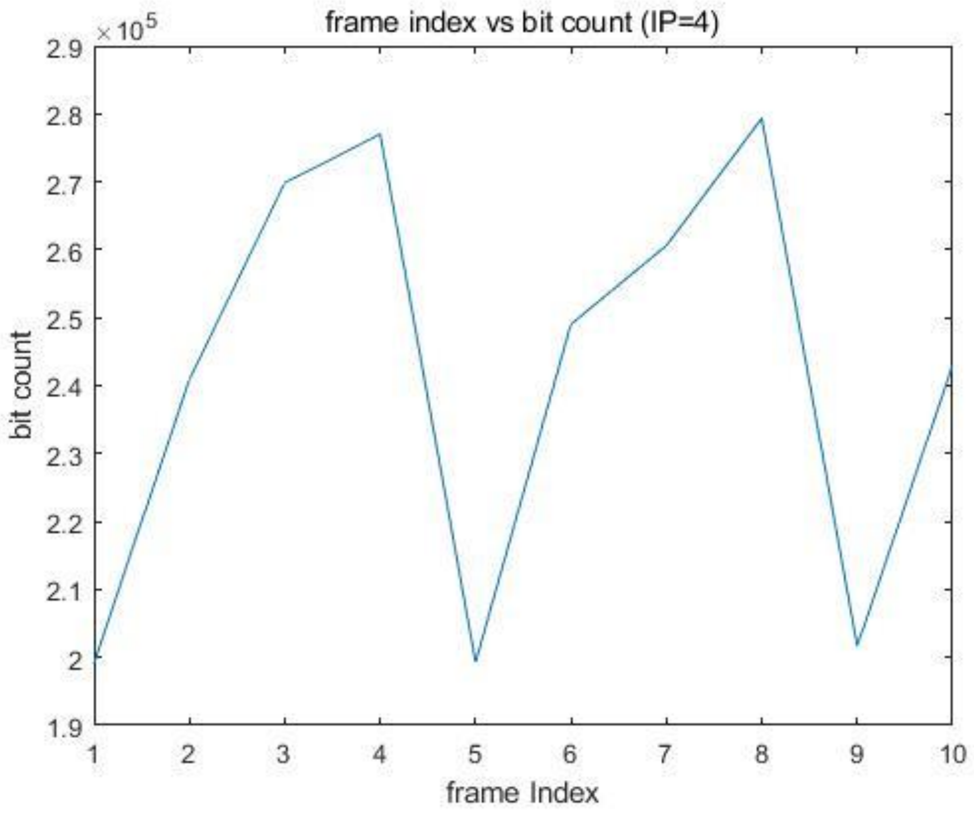
2. bit-count vs. frame index

(i) With constant I =8 r = 2 and QP = 3, the bit count (x-axis) vs frame index (y-axis) with different I_Period value has been plotted below (All the result are generated by only running first 10 frames of the video):

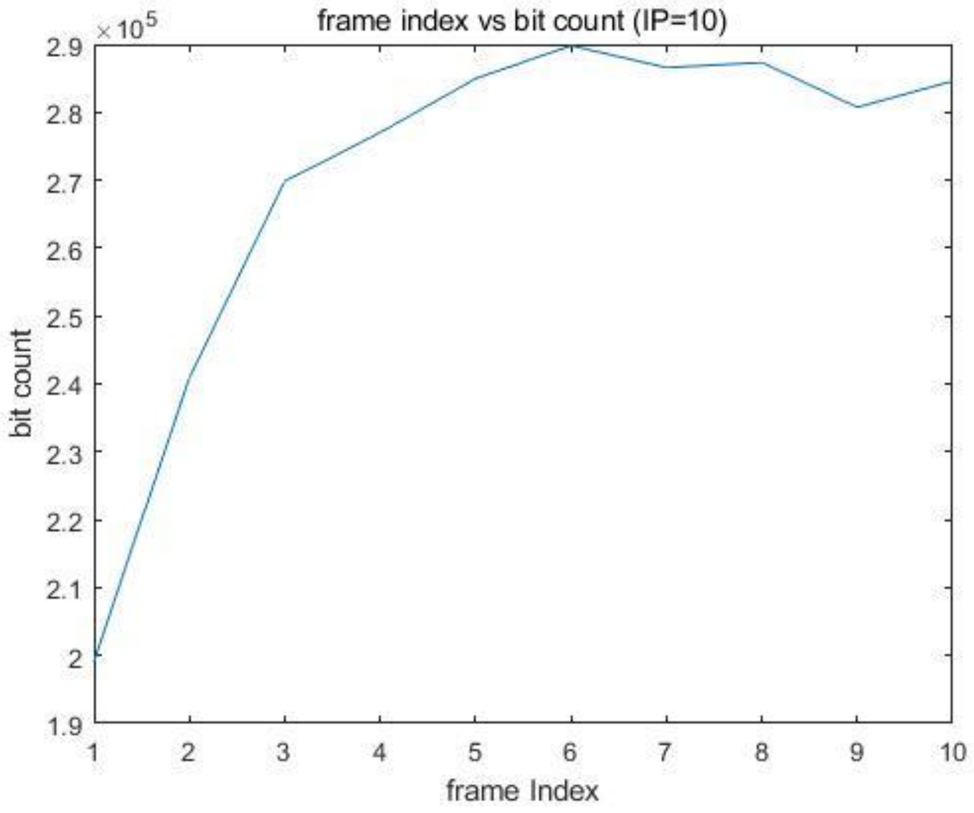
I_Period = 1:



I_Period = 4:

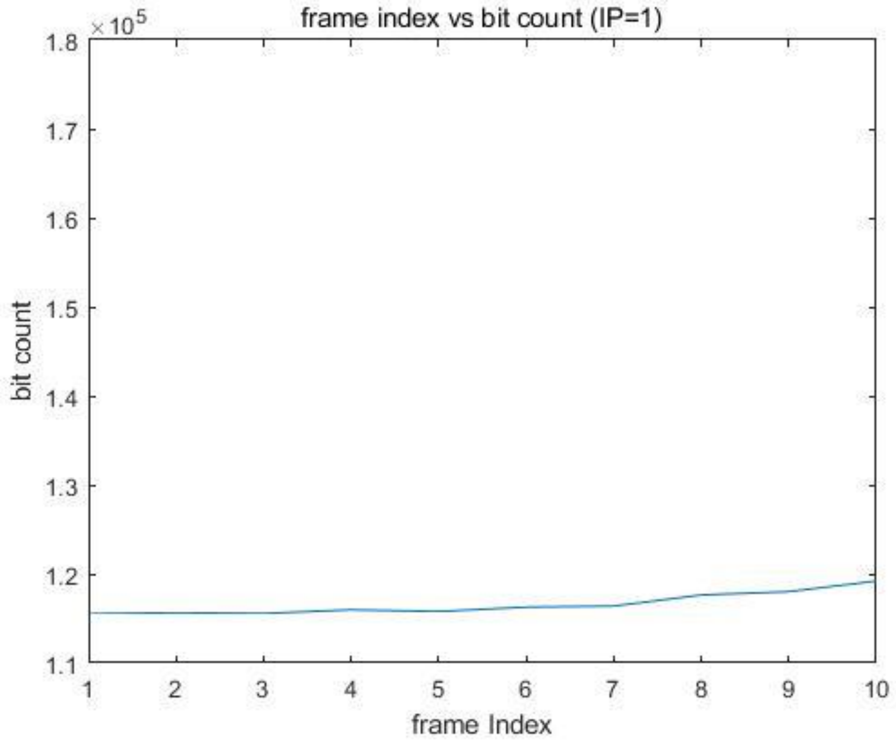


I_Period = 10:

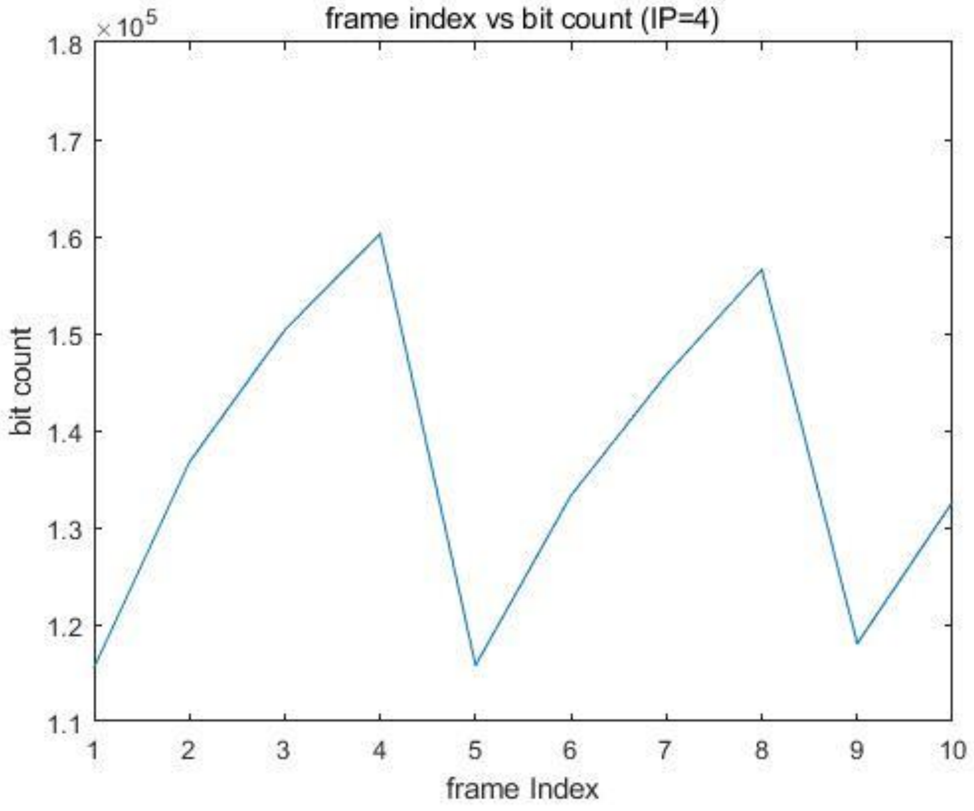


(i) With constant I =16 r = 2 and QP = 3, the bit count (x-axis) vs frame index (y-axis) with different I_Period value has been plotted below:

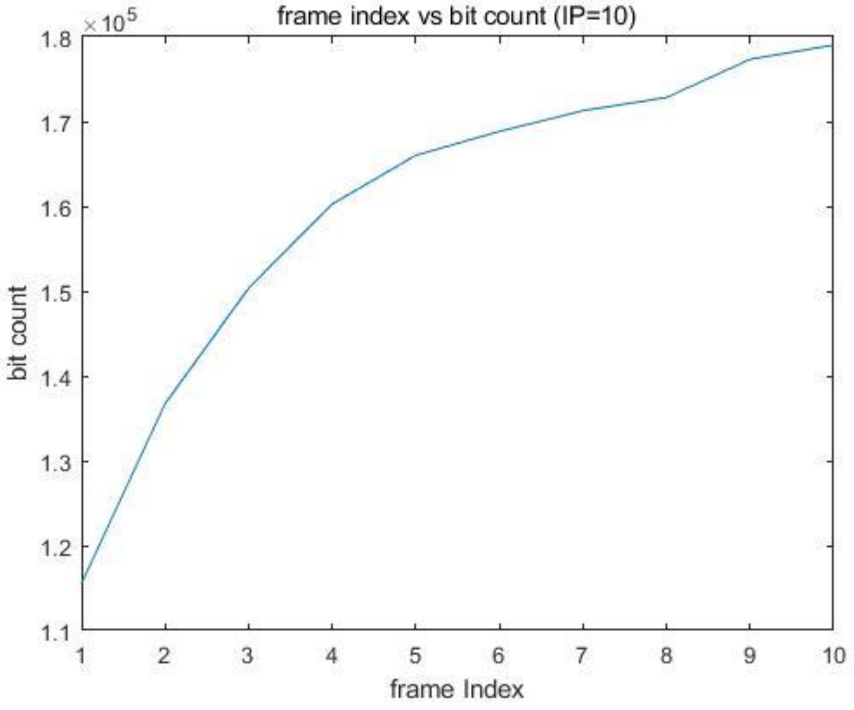
I_Period = 1:



I_Period = 4:



I_Period = 10:



3. **Which kind of frames (Intra or Inter) typically consume more bitrate? Why? Is this true across the entire QP range?**
   The Inter Frame takes more bitrate. As shown in the plots from exercise part 2, the inter Frame always take more bitrate than Intra Frame. Though an image have bits that are all different from one another, during intra prediction a block will be filled with repetitive values. After undergoing RLE, the repetitive values is compressed more thus providing lesser bits of information to transmit.

4. **For (i=8, search range = 2, and QP=3), what is the compression ratio of the 10 frames of Foreman CIF compared to the uncompressed Y component? What is the average PSNR?**
   The following tables shows our result

	PSNR	Time	**Total Number of Bits for 10 frames for compressed video**	Total Number of Bits for 10 frames for orignal video	**Compression Ratio**
IP = 1	**-129.39**	879	**2002194**	1022400	**0.5106**
IP = 4	**-138.87**	789	**2419752**	1022400	**0.4225**
IP = 10	**-140**	1143.1	**2487675**	1022400	**0.4110**

As you can observed from table, for QP=3, we are actually achieving negative compression ratio. This is because the QP value is too small. The average PSNR is -136.1.

Conclusion

The encoder engine and decoder engine has been implemented in Matlab with the requirement from the assignment instruction. There are some obvious relation can be observed between parameters and result(execution time and quality of decoded video). For example, large r will lead higher execution time. Low QP will result better result but with higher execution time and low compression ratio.