Project 2 – H.264-based Motion Compensation Residual Coding

Yang Deng 450616938 Qing Xu 460129169

1.Objective

1. investigate and implement the H.264/AVC 4×4 integer DCT algorithm.

2. replace division by bit shift operation in quantization.

3. implement H.264/AVC inter frame coding.

4. compare performance between integer DCT algorithm and general DCT algorithm in terms of time complexity and PSNR.

2.Introduction

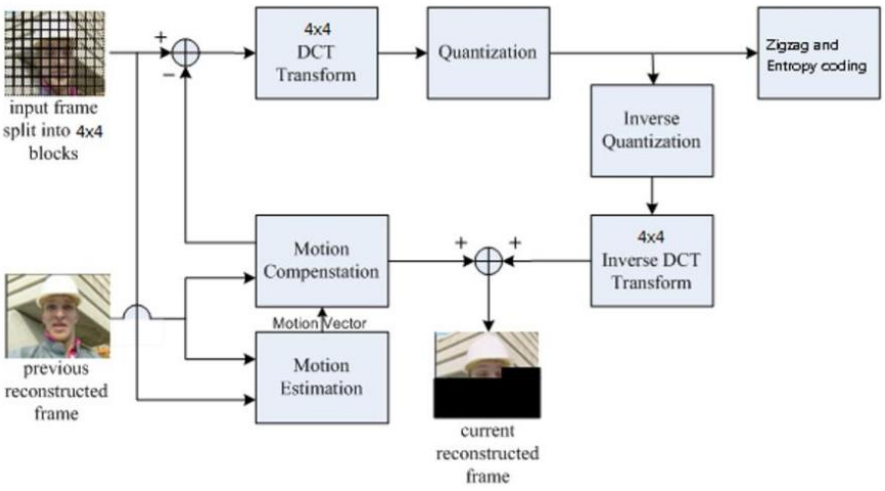


Figure1 the framework of inter frame coding

In order to reduce time complexity, H.264/AVC applies 4by4 Integer DCT algorithm which replaces all multiplications by divisions and subtractions. Also, it combines the DCT coefficient with quantization step, transforming these operations to a integer multiplication and bit shift operation. This report will introduce the principle of the methods, how to implement the methods and discuss the performance contrast between new methods and old ones. Fig1 shows the whole inter frame coding flow

chart. Each frame is split into 4by4 blocks and all the DCT and quantization as well as encoding is based on these blocks. Motion Estimation and Motion Compensation is performed by the given function FSBMAI. Additionally, the entropy coding is given by expgolomb.m. Our task is to implement the rest of parts in the framework.

3.Methodology

3.1 Fast 4 by 4 DCT

The general 4 by 4 DCT and IDCT is given by

and

$$
\begin{aligned}
\mathrm{Y}' &= (\mathrm{C_f X C_f}^{\mathrm{T}}) \otimes E_f \\
&= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
Y &= \mathrm{C_i}^{\mathrm{T}}(Y \otimes E_i)\mathrm{C_i} \\
&= \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(X \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}
\end{aligned}
$$

where a=$\frac{1}{2}$ and b =$\sqrt{\frac{2}{5}}$.**Leave the Ef and Ei to the quantization and dequantization** , the rest of matrix calculation could replace by simply multiple and addition.

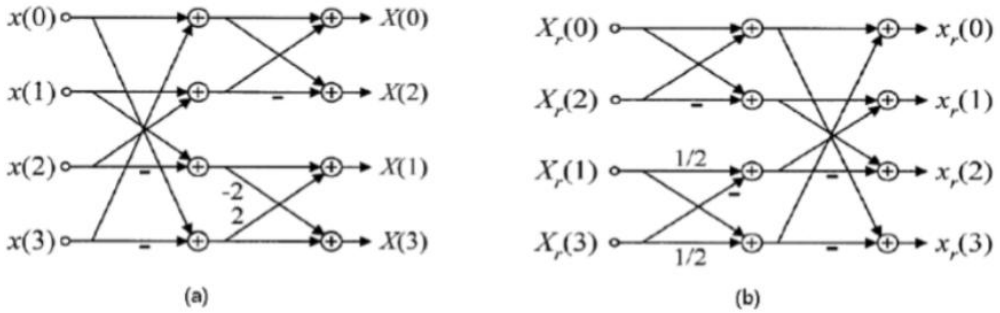


Figure2 implementation of fast 4 by 4 DCT and IDCT

3.2 Quantization

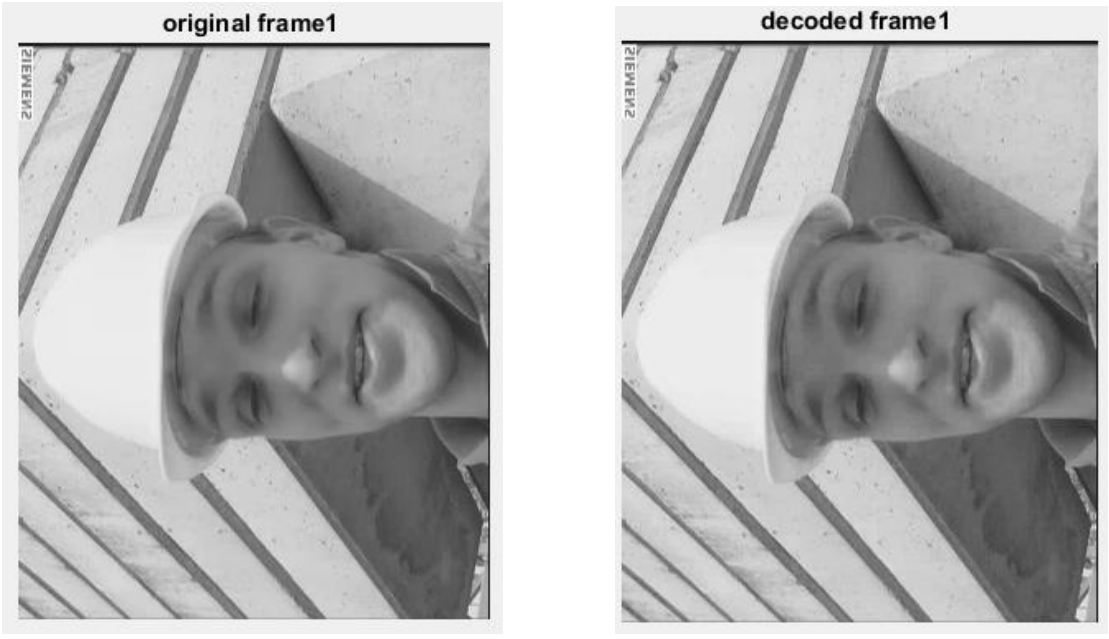
H.264/AVC combines the DCT remained Ef with quantization step.

$$quantized\_coefficient = (C_f X C_f^T) \otimes (\frac{PF}{Q_{step}})$$

To simplify the calculation, $\frac{PF}{Q_{step}}$ is replaced by $\frac{MF}{2^{Qbits}}$. If we right shift a integer MF Qbits, the same operation could be given by $floor(\frac{MF}{2^{Qbits}})$. Therefore, we can replace the division operation by right shift in this case.

4.Simulation Results

4.1 Original & encoded frames



(a) **(b)**

(c) **(d)**

(e) **(f)**

Figure3 the 1st, 5th,and 10th frame of original video and decoded frame

As shown in figure3, (a),(c),(e) shows the original video frames and (b),(d),(f) shows the reconstructed video frames under condition that **QP(intra) = 22, QP(inter) = 24**. and **search range = 8 by 8**. We can see the first decode frame is relatively close to the original frame, while the other two decoded frame is relatively burred and we can subtly see there are 4 by 4 patches on the image.

4.2 Performances of encoder

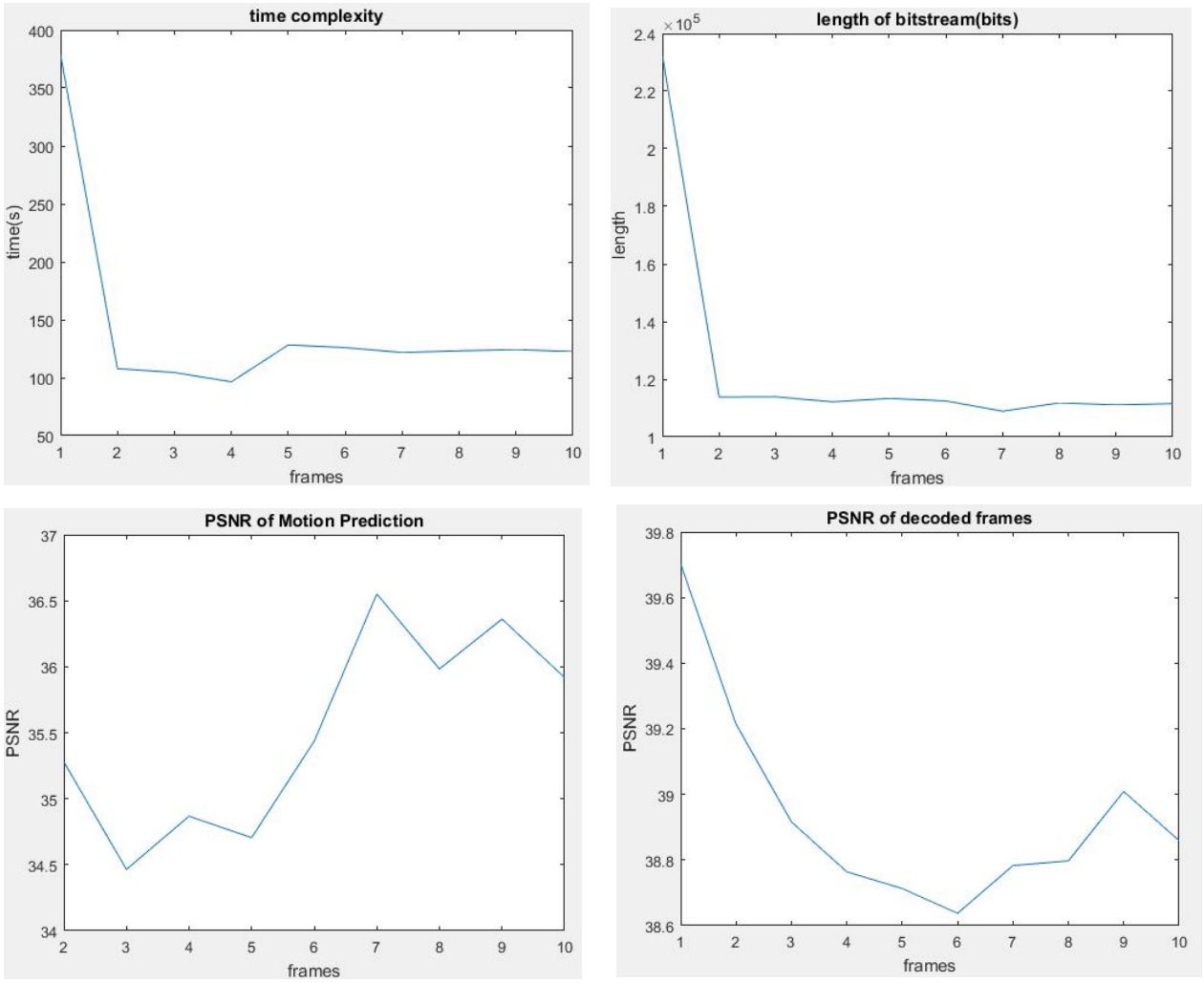


Figure4 search range = 8 * 8, QP(intra) = 22,QP(inter) = 24

Table1: simulation results in details

frame	time (s)	length	PSNR_ME	PSNR_frame
1	378.88	232434	**NA**	39.70
2	107.81	113812	35.28	39.21
3	104.49	113912	34.46	38.92
4	96.41	112142	34.87	38.76
5	128.27	113342	34.70	38.71
6	125.95	112466	35.44	38.64
7	121.71	108896	36.55	38.78
8	123.18	111798	35.98	38.80
9	124.26	111082	36.36	39.01
10	122.49	111534	35.92	38.86

Fig4 and table1 show the performance of the encoder in terms of time complexity, length of the code, PSNR of predicted frames, and PSNR of encoded frames. Since

the first frame is set as the I frame and other frames are P frames, the performance of first frame has a obvious difference from other frames. Table 2 shows the difference.

Table2 Performance difference between I frame and P frames

	First frame (I frame)	Other frames (P frame)
time	378s	**One third** of first frame
length	232kbits	**Half** of the first frame
PSNR_ME	Inf or NA	Around 35
PSNR_frame	39.7	Around 39

The reason of the difference is that I frame is intra-coding where needs more bits to record self information, while P frame is inter-coding where part of information is saved in its previous frame. Also, after DCT, the residual frame of P frame contains more 0s, so the code length could be reduced. The total code length of the 10 frames video is 1241418 bits. Given the original YUV file is 1485KB, so the compression ratio is $\frac{1241418\text{bits}}{1485000*8\text{bits}}=1:9.57$. Since when QP = 24, the quantization step is 10, this practical compression ratio relatively verifies the theory. The 0.5 loss of compression ration comes from the first frame which takes twice times of bits than other frames to transmit.

4.3 Performance of fast DCT & Quantization

In order to test the performance of the DCT and quantization exclusively, the first 4 by 4 block of first frame will be performed DCT transform & quantization process under the condition that QP = 17 to 22 respectively.

Original block :

8	11	10	6
42	38	37	41
201	200	205	212
253	255	251	245

After DCT & Quantization

112	0	0	0
−91	0	0	0
3	1	−1	0
12	−1	0	0

QP = 17

100	0	0	0
−81	0	0	0
3	1	−1	0
11	−1	0	0

QP = 18

91	0	0	0
−74	0	0	0
3	1	−1	0
10	−1	0	0

QP =19

77	0	0	0
−63	0	0	0
2	1	−1	0
8	0	0	0

QP = 20

72	0	0	0
−58	0	0	0
2	1	0	0
8	0	0	0

QP = 21

63	0	0	0
−51	0	0	0
2	1	0	0
7	0	0	0

QP = 22

After iDCT & Dequantization

8	10	9	7
42	40	39	42
202	202	205	209
252	253	250	247

QP = 17

9	11	10	8
41	39	38	41
200	200	203	208
249	251	248	244

QP = 18

9	11	10	7
41	38	38	40
200	199	203	208
251	252	249	245

QP =19

8	10	8	3
40	38	40	44
203	200	202	207
250	252	250	246

QP = 20

12	11	9	8
38	39	42	43
202	203	206	207
251	250	248	247

QP = 21

12	11	9	7
37	39	41	42
202	203	205	207
253	251	249	248

QP = 22

Table 3 4 by 4 block PSNR when QP is from 17 to 22

QP	17	18	19	20	21	22
PSNR	44.6090	41.6594	43.7375	40.4407	38.5583	39.2029

Based on the results, we can see as the QP increases, namely the quantization step increase, there are more 0s appearing at the quantized blocks. (The yellow highlight indicates the new zero). As for PSNR, it approves that PSNR is not strictly getting lower when QP is going up. It is also block related, but the total trend definitely shows the that greater quantization step will result in small PSNR. More details about QP and PSNR is discussed in section 4.2.

4.4 Time distribution

Form section 3.3, we can see it takes around 1433s(23mins) to encode the video and about 120s to encode a P frame. In this section. We will find out the most time-consuming part of the encoder. The experiment parameters are **QP(intra) = 22, QP(inter) = 24, and 8 by 8 search range**.

Results :

Table 4 time distribution of encoder(unit : s)

frame	MP	DCTQ	coding	IDCTQ	others	total
1	0.00	0.17	378.58	0.14	0.0015	378.88
2	1.15	0.16	105.36	0.14	1.0015	107.81
3	1.19	0.16	101.01	0.13	2.0015	104.49
4	1.17	0.14	91.97	0.13	3.0015	96.41
5	1.09	0.14	122.91	0.13	4.0015	128.27
6	1.11	0.14	119.57	0.13	5.0015	125.95
7	1.17	0.15	114.27	0.13	6.0015	121.71
8	1.20	0.15	114.69	0.15	7.0015	123.18
9	1.20	0.14	114.79	0.13	8.0015	124.26
10	1.10	0.18	112.07	0.14	9.0015	122.49
total	10.38	1.53	1375.19	1.33	45.01	1433.44
%	0.72%	0.11%	95.94%	0.09%	3.14%	100.00%

Table 4 shows the time spending on every part of encoder, where MP denotes the

motion prediction. It is obvious that the coding account for the most of the time(95.4%). we may have to upgrade the coding algorithm to reduce the encoding time.

5. Discussion

There are two factors that affects the performance of the encoder, including search range and QP.

5.1 factor 1 - Search Range

The search range experiment is based on the condition that **QP(intra) = 22**, and **QP(inter) = 24**. The results are as follows:

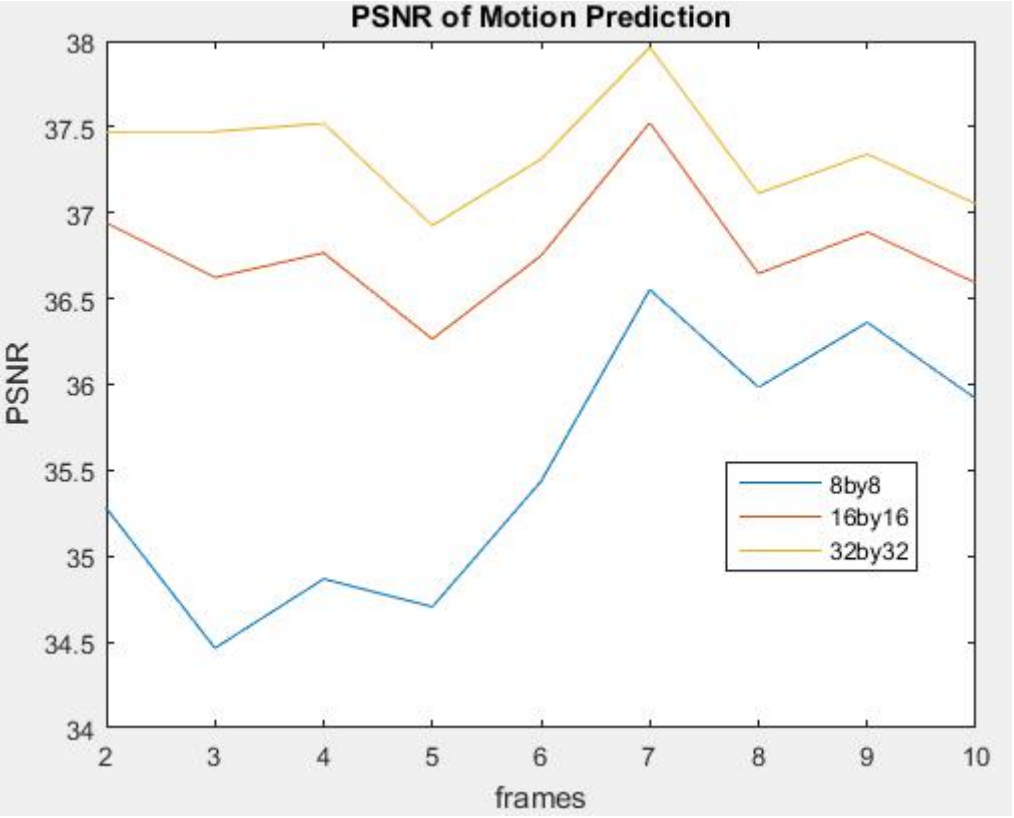


Figure5 PSNR of predicted frames with different search range

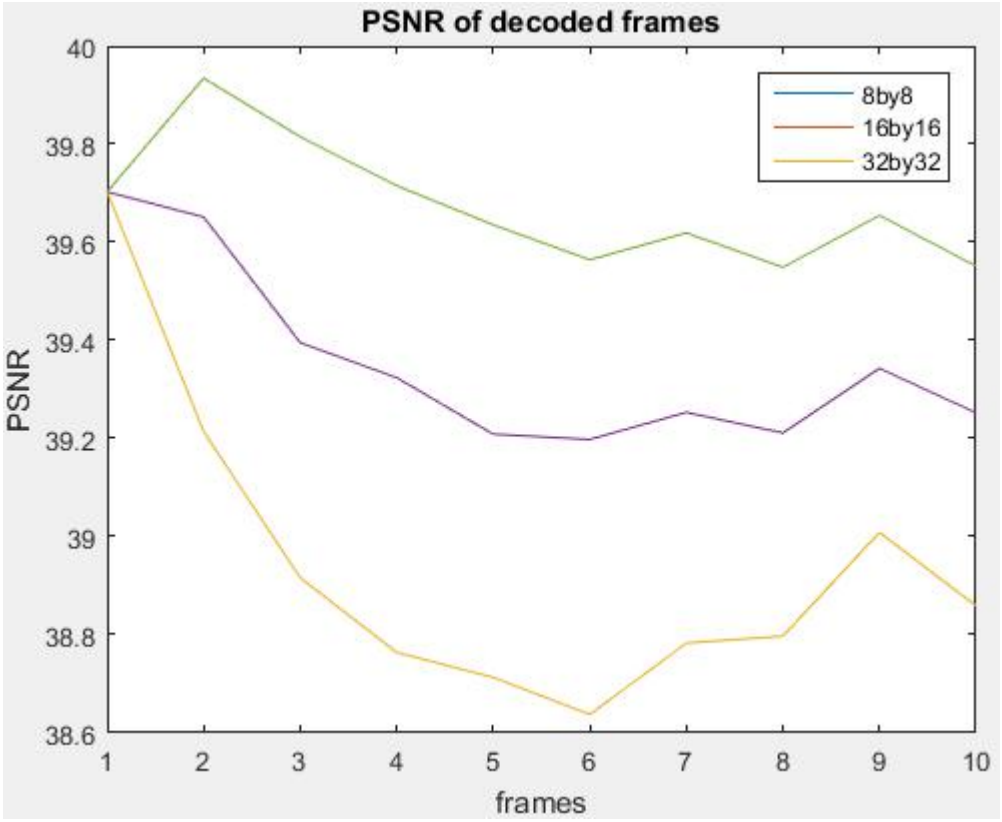


Figure6 PSNR of predicted frame with different search range

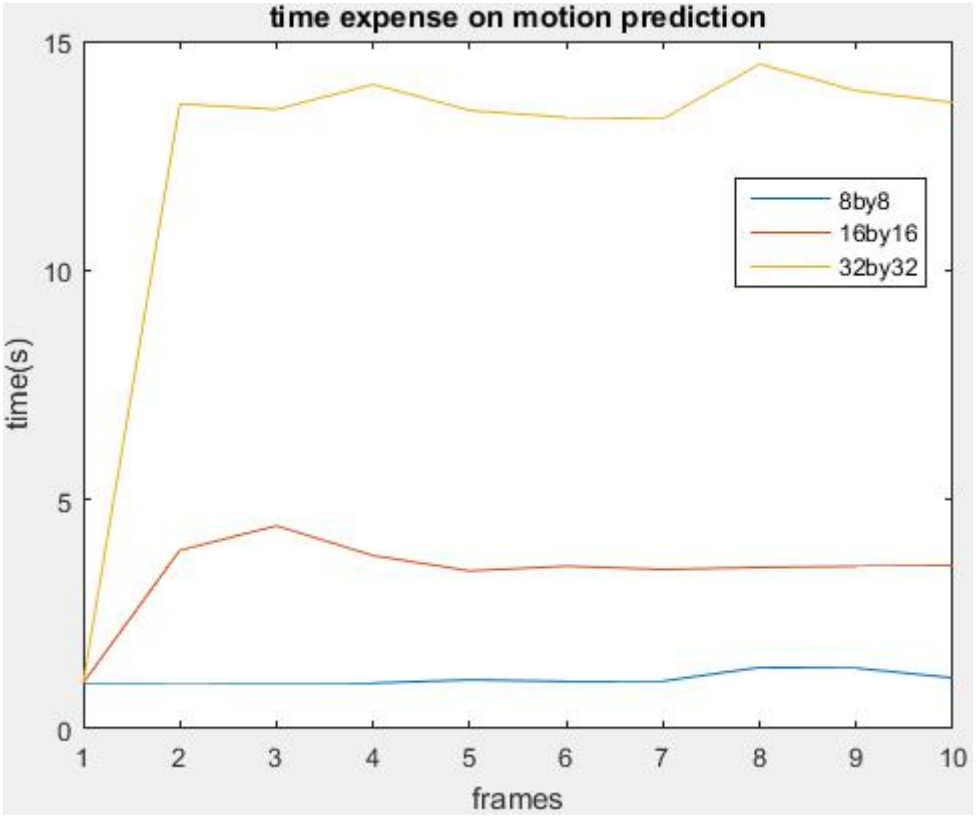


Figure7 time expense on motion prediction with different search range

The performance shown in fig 5,6&7 can be summarized that larger searching range

results in more time-consumption and higher PSNR both of predicted frames and reconstructed frames. It is worth to point out that four times searching range expansion will lead to four times searching time but less one PSNR increase, therefore it is not worth to expand searching range from 8 by 8 window.

5.2 factor2 - QP

The Quantization step experiment is based on the condition that **search range = 8 by 8 block and QP(intra) = 22.**

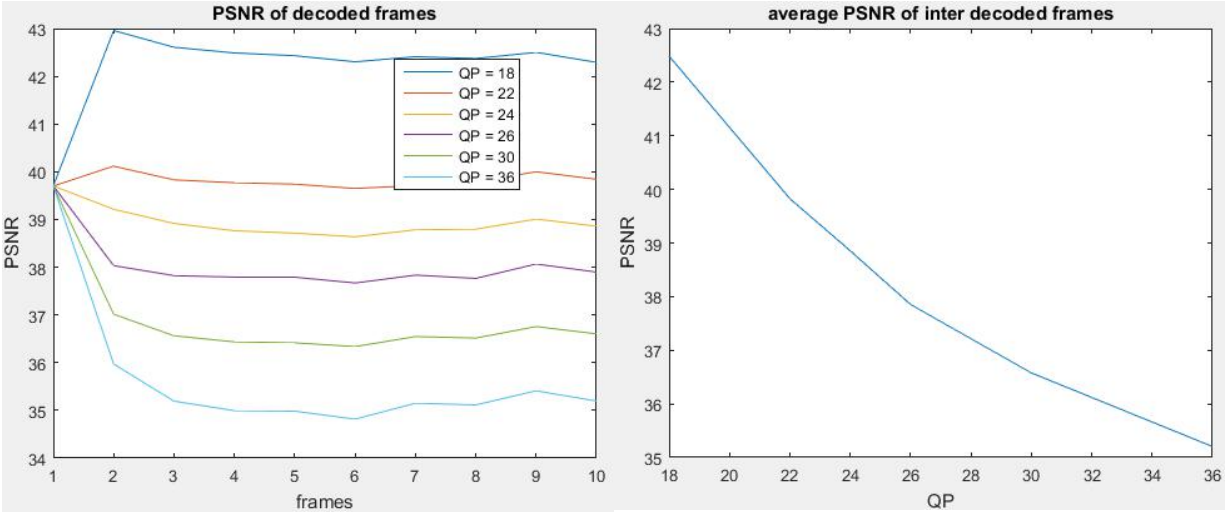


Figure8 results of varying QP experiment

Fig8 shows the PSNR of reconstructed frames with QP = 18,22,24,26,30, and 36 respectively. It shows that larger quantization step results in lower PSNR and the relationship of them can be seen as linear in a small window(at least from 18 to 30).

6. Conclusion

This project implements the encoder of YUV420 video. I frame consumes more time and bits to record but along with higher quality, while P frames have opposite properties. In this simulated encoder, the expgolomb coding accounts 95% time and could use an upgrade. Enlarging QP could get better compression ration but lower

video quality. There should be a very careful trade-off during the design. Also, the *8 by 8 search window has the best cost-performance ratio among 8,16 and 32 window ranges.

7. Contribution

	Yang Deng 450616938	**Qing Xu 460129169**
coding	**Encoder.m** **DCTQ.m** **iDCTQ.m** **Paras.m** **expgolomb.m(modify)** **YUV_read.m**	**Discussion.m** **Evaluate.m** **PSNR.m** **Show_img.m** **ZigZag.m**
report	**Methodology** **Simulation Results** **Conclusion**	**Objective** **Introduction** **Discussion**