Signal Processing

Coursework Project Report

Autumn 2019

Name: Haibo Lei.

Student ID: 1910273011

1. Introduction

Motion estimation (ME) is crucial to the performance of a video codec as it recognizes and mitigates temporal redundancies between consecutive frames of a video sequence. It is one of the most computationally intensive blocks in the video encoder. Block matching algorithm (BMA) is considered as one of the best approaches for performing ME due to its ease of implementation and efficiency [1]. Therefore, it is necessary to find a good block matching algorithm.

In this task, we are required to use the given two video sequences as input and implement several programs that reads frames from YUV files and performs block wise motion estimation on 16x16 pixel blocks. Finally, compare their performance in terms of accuracy and complexity.

2. Methods

In this project, we use python to implement a YUV reader[2] and three motion estimation methods: full search, three step search, and diamond search[3]. What's more, I made some modifications to the three-step search method to increase the number of search steps, and I call it multi step search.

2.1 Full Search

Full search is a complex brute force approach. It requires the program to traverse each point based on the size of the window to find the best match point. However, it is the easiest method to implement and can be used as a baseline. The algorithm flow is shown in Algorithm 1[4].

Algorithm 1. Full Search

Algorithm 1 FullSearch.

```
param fr: the frame need to predict
param fr_ref: previous frame
param window_size: full search window size

Function FullSearch(fr, fr_ref, window_size)
  for each block in fr:
    move the block in search area of fr_ref:  #base on window_size and block location
      calculate error()
    find the best match block
    replace the fr_block use fr_ref_block
```

Algorithm 2. Three Step Search

Algorithm 2 ThreeStepSearch.

```
Function ThreeStepSearch(fr, fr_ref)
  for each block in fr:
    original_point = center of block
    for S in [4,2,1]:
      points = get search point(S) #8 locations +/- S pixels around original point and the original point
      for each point in points:
        calculate error()
    original_point  = the minimum cost point
    # replace
    original_point is the center of best match block
    replace the fr_block use fr_ref_block
```

2.2 Three Step Search

The three-step search method only needs to search at certain key points, which greatly speeds up the search efficiency, but this algorithm may not find the best matching point. The algorithm flow is shown in Algorithm 2.

2.3 Diamond Search

Diamond Search algorithm uses a diamond search point pattern, and it is non-linear algorithm. However, for most videos it always works well. The algorithm flow is shown in Algorithm 3.

Algorithm 3. Diamond Search

Algorithm 3 DiamondSearch.

```
Function DiamondSearch(fr, fr_ref)
  for each block in fr:
    original_point = center of block
    LDSP:
      S = 2
      search 9 locations pixels (X,Y): #(|X|+|Y|=S) around location original_point and original_point
        calculate error()
      original_point  = the minimum cost point
      if original_point is found at center of search window:
        goto SDSP
      else:
        goto LDSP
    SDSP:
      S = 1
      search 5 locations pixels (X,Y):
        calculate error()
      original_point  = the minimum cost point
    # replace
    original_point is the center of best match block
    replace the fr_block use fr_ref_block
```

Algorithm 4. Multi Step Search

Algorithm 4 MultiStepSearch.

```
Function MultiStepSearch(fr, fr_ref, step)
  for each block in fr:
    original_point = center of block
    for S in get_S(step):      # S = [2^(step-1),2^(step-2), ..., 8,4,2,1]
      points = get_search_point(S) #8 locations +/- S pixels around original point and the original point
      for each point in points:
        calculate error()
    original_point  = the minimum cost point
    # replace
    original_point is the center of best match block
    replace the fr_block use fr_ref_block
```

2.4 Multi-Step Search

The Multi-step search method is based on three-step search method. I just increase the number of search steps to adapt more rapidly changing video, while only adding linear time cost. The algorithm flow is shown in Algorithm 4. In this function, if step = 3, it will become three-step search.

3. Experiment

3.1 Pre-experiments

For Full Search method, prediction error and complexity are related to search window size. To avoid spending too much time on the calculations, we performed some pre-experiments to determine reasonable parameters.

We used the first frame of dragon video to predict the second frame, test with different search window sizes, and results are shown in Fig.1.

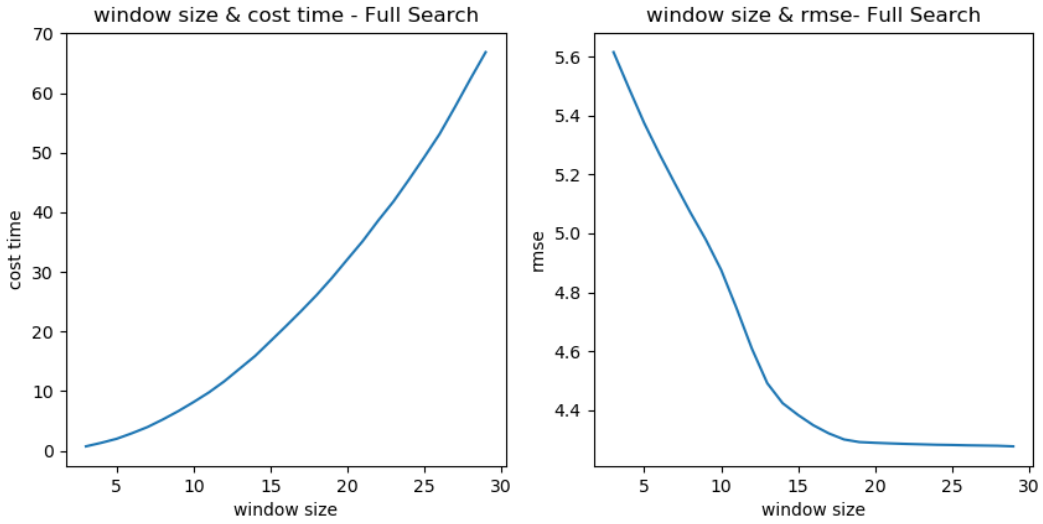

Figure 1. Different search window size on Full Search method

Fig.1. shows that for Full Search method, if we use larger window size, we can get a lower error rate, but the time it takes will increase explosively. Considering the time cost, we finally chose 5 as window size.

We plot the predicted result in dragon video use Full Search when window size is 20, and results are shown in Fig.2.

Figure 2. Predicted result in dragon video left: original mid: predicted right: difference

3.2 Experimental Results and Analysis

We used four methods to test each videos, and the test process is shown in Fig.3.

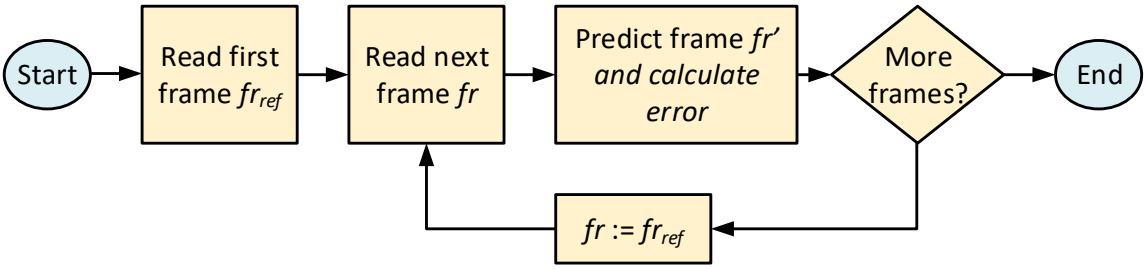

Figure 3. flow chart for main program

Finally, the results obtained by our program are shown in Table 1.

Table 1. Example table caption.

	Video 1 (dragon) RMSE/psnr/time	Video 2 (gas) RMSE/psnr/time	Average RMSE/psnr/time
Full search (window size=5)	5.691/33.0/190	1.267/47.1/190	3.479/40.0/190
Three-step search	5.561/33.2/60	1.218/47.2/60	3.389/40.2/60
Diamond search	5.540/33.3/49	1.188/47.5/34	3.364/40.4/42
Multi-step search (step = 4)	5.287/33.7/80	1.289/46.4/80	3.288/40.1/80

Comparing those result, we can find that the diamond search method is the only non-linear search method, which has different performances for different videos. If the changes between each frame are large, it takes a long time to calculate , otherwise it needs a shorter time. For dragon video, which changes between each frame are large, Multi-step search (step = 4) gets the lowest RMSE, because it has a largest search area. However, for gas video, Diamond search perform better. In general, of these four algorithms, Diamond search performs best, because it not only has lower RMSE, but also has the lowest complexity, which can be completed in almost half the time of other algorithms

4. References

[1] Rohan Mukherjee, Indubu Gaana Vinod, Indrajit Chakrabarti, Pranab Kumar Dutta, Ajoy Ku mar Ray:Hexagon Based Compressed Diamond Algorithm for motion estimation and its dedicate d VLSI system for HD videos. Expert Syst. Appl. 141 (2020)

[2] Python 读取 YUV(NV12) 视频文件实例子. http://www.45fan.com/article.php?aid=191209 75079715808750127559.

[3] Zhu, Shan; Ma, Kai-Kuang (February 2000). "A New Diamond Search Algorithm for Fast Bl ock-Matching Motion Estimation". EEE Trans. Image Processing. 9 (12): 287–290.

[4] Wikipedia. Block-matching algorithm. https://en.wikipedia.org/wiki/Block-matching_algorith m#cite_note-7.