Experimenting with Video Retrieval

Project Phase II Report

Submitted By

Group 19

DARSHAN D SHETTY (1211169970)

BHARATH KUMAR SURESH (1211182086)

UTHARA KEERTHI (1211106023)

DHANANJAYAN SANTHANAKRISHNAN (1211181423)

SHIVAM GAUR (1211181722)

Abstract

Video similarity measurement involves matching temporal and spatial metrics for objects in the video. Since similarity determination depends on the subjectivity of the user, the procedure is inherently complex as it requires considerations of not only the content of the video but the needs of the user as well. The goal of this phase of the project is to determine similarity between videos considering features that were previously extracted in Phase 1 (Color Histogram, SIFT vectors, Motion vectors). After determining similarity measures for videos based on these features, for a given video and a set of frames in the video, their similarity to other videos in the database is determined. This similarity determination does not consider user subjectivity to filter results but the results, however, are evaluated to simulate visual perception as close as possible. For large number of features, dimensionality curse is the greatest concern. To avoid this and demonstrate dimensionality reduction for the current set of features, PCA and K-means are used. After reducing dimensions for the current set of inputs, the video similarity is determined again in order to demonstrate that the reduced number of dimensions can also be used to determine similarity effectively and efficiently.

Keywords: Color Histogram, SIFT, motion vector, video similarity, PCA, K-means

Introduction

Ø Terminology

Color Histogram of a digital image contains information about the number of pixels of a color present in the image's color space. Color histograms can also be used for image retrieval [16].

SIFT (Scale-invariant feature transform) is an algorithm used to identify local features in the image. This is used to obtain points of interest in the image. This information can be used to locate objects of interest in the image [17].

Motion Vector describes how an image transforms into another, usually for frames next to each other in a video. Motion vectors are used in motion estimation for determining how objects move from one image to other. This is very useful in reducing the amount of information needed to describe the motion of objects and hence applied extensively in video compression [18].

Intersection Similarity also called Jaccard Index/Similarity Coefficient is a measure of the of how similar two distributions are. It compares the maximum and minimum value of the distribution to determine how close two data sets really are[11].

$$sim_{int}\left(\vec{a},\vec{b}\right) = \frac{\sum_{i=1..n}\min(a_i,b_i)}{\sum_{i=1..n}\max(a_i,b_i)}$$

Cosine Similarity is an angle similarity measure which is designated to measure the similarity between a set of dynamic data sets according to their inner product space. It is generally implemented in positive part of statistical space and the outcome in this situation is bounded between 0 and 1 and is heavily used for text based comparisons. Its major use is in data mining and information retrieval [12].

$$sim_{cos}\left(\vec{a},\vec{b}\right) = cos\left(\vec{a},\vec{b}\right) = \frac{\vec{a}\,.\,\vec{b}}{|\vec{a}||\vec{b}|}$$

Mahalonobis Distance is a measure of distance between two vectors in a vector space which denotes the analysis of their Euclidian distance along with an emphasis on Statistical Distribution of points in space. Also, it can be concluded as a multi-dimensional approach towards standard deviations calculations[13].

$$\Delta_{Mah}\left(\vec{a},\vec{b}\right) = \sqrt{(\vec{a}-\vec{b})^T\, S^{-1}\,(\vec{a}-\vec{b})}\ \ where\ S[x,y] = Cov(x,y) = E((x-\mu_x)(y-\mu_y))$$

L_1 Distance is based on Taxicab geometry, in which the distances between two considerable data points is taken as a summation of their absolute distances, also known as L_1 norm or Manhattan distance; which is a specific case of Minkowksi metrics[14].

$$L_1 = \sum_{i=1..n} d_i$$

Hungarian Algorithm is an optimization algorithm that use combinatorics to solve designated problem in polynomial time. It is used majorly for assignment problems and further led to primal–dual methods. Time complexity is $O(n^4)$ and can also be modified to achieve $O(n^3)$. Also called, Munkres[1][6] assignment heavily used to match assigned fields perfectly and in minimum costs. Inferred for Bipartite graph and array implementation[4].

Stable Marriage Algorithm compares two equal size data sets and gives a perfect matching; under the order of preference provided. In a large data set of dynamic content it can be used for assignment operations[5].

Principal Component Analysis also called K-L transform is a statistical procedure to convert a set of correlated variables distributed and culminated in a set relation into a set of single/linear related variables. The procedure is heavily used to reduce dimensionality of a vector space under consideration. It uses a covariance matrix relation and eigen decomposition relations to obtain a final set of vectors which are present in a reduced or possible linear dimension. Also, a principal component is one which provides no advantage if a further dimensionality reduction

is done because it relates to all features symmetrically. This approach is very sensitive to original description of vector space[9][15].

K-means clustering is a popular method for cluster analysis in data mining. This technique is used for quantizing vectors in their original space to obtain a reduced dimensionality space representation. It partitions n objects or n input vector space points and segregate them into k clusters. [3][10]

Ø Goal Description

TASK #1

Given the output data generated from the video files in Phase1 for Color Histogram, SIFT vectors and Motion vectors, the goal is to implement a program which computes the Color Histogram similarity, SIFT similarity and Motion similarity for two given video files.

TASK #2

Given the output data generated from the video files in Phase1 for Color Histogram, SIFT vectors and Motion vectors, the goal is to implement a program - for a given video file and a given frame range – which returns k most similar frame sequences and visualizes the query and results as videos.

TASK #3

Given the output data generated from the video files in Phase1 for Color Histogram, SIFT vectors and Motion vectors, the goal is to implement a program which reduces the dimension of the vector space to a given dimension d using PCA and K-Means and outputs the d dimensions in terms of the input vector space.

TASK #4

Given the output data generated from the video files in Phase1 for Color Histogram, SIFT vectors and Motion vectors, the goal is to implement a program - for a given video file and a given frame range – which returns k most similar frame sequences using the dimensionality reduced vector space obtained from Task #3 and visualizes the query and results as videos.

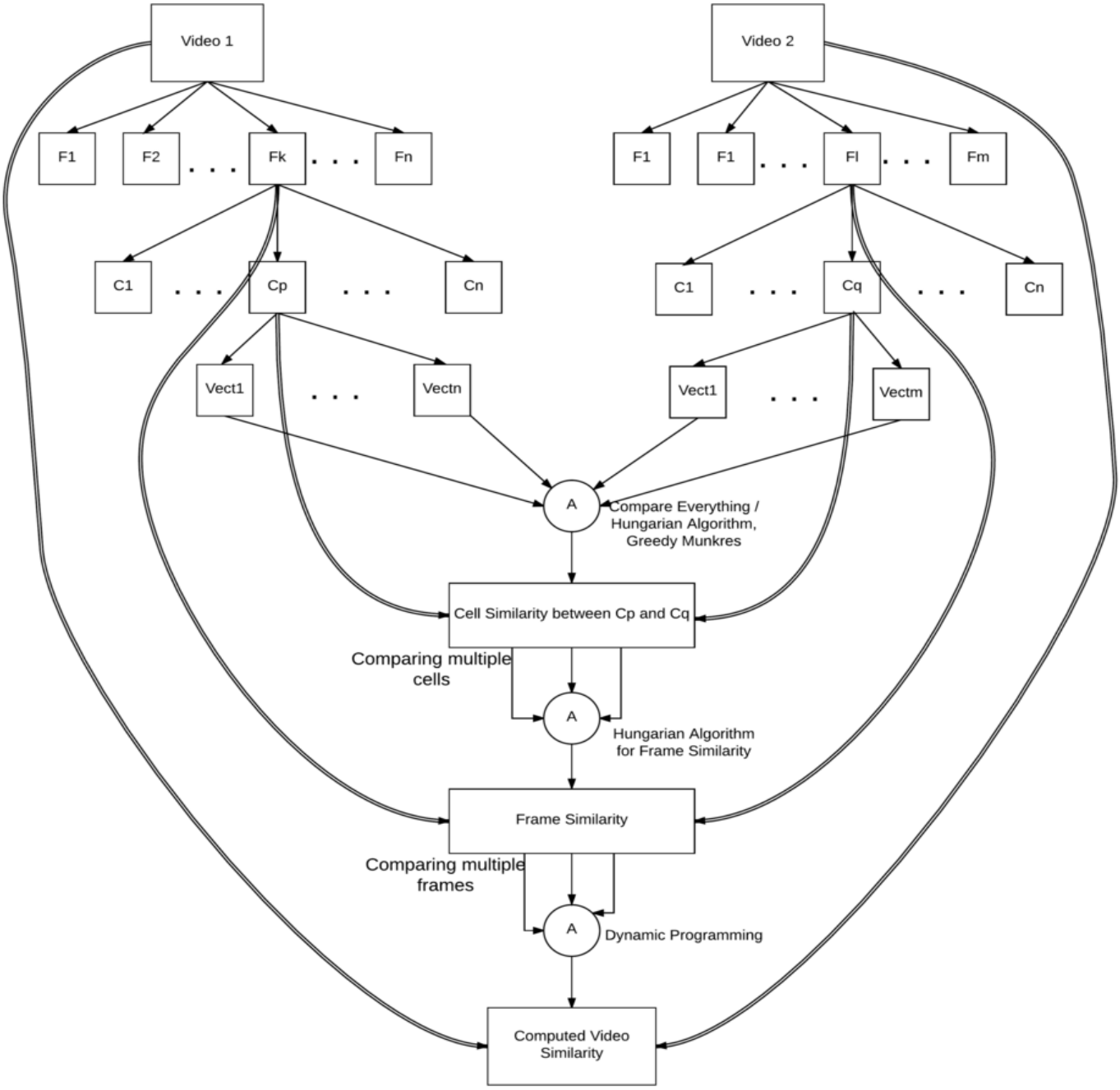
Ø Assumptions

- The path to the input files is already known. The file names are out_file.chst for Color Histogram, out_file.sift for SIFT vectors and out_file.mvect for Motion vectors.
- The Phase 2 file inputs are of the form mentioned in Phase 1. Also, memory is not a huge concern for the user as the data structure which stores the input data depends on the number of videos and hence, may be large.
- The programs process videos sorted by their names in alphabetical order. The video index starts from 0. i.e. Video 0 corresponds to the first video and video n-1 corresponds to the last one.
- The input video files are assumed to be MP4 files.
- Frame subsequence range [a,b] given for a particular video (for task #2) lie within the frame range for that video i.e. from [0,no. of frames in that video -1].
- The 'k' most similar subsequences returned have frame lengths greater than or equal to the length subsequence range [a,b] given in the query.
- If two subsequences match with the same similarity/distance, precedence is given to one arbitrarily.
- All the 'k' most similar subsequences are from different videos present in the directory, i.e. only one subsequence from a video is matched. This implies the value of k should not exceed the number of videos present in the directory.
- For K-Means, the number of dimensions is taken as the number of clusters.
- For Task 3, after reducing the dimensions, the columns represent the new dimensions, the rows correspond to observation or data and $v_1, v_2, ..., v_d$ are the vectors in the dimensionality reduced vector space.
- SIFT vector and Motion vector similarity/distance measures takes a long time to compute since the input fetched is large in size. So it is assumed that time complexity is of less importance to user.
- Also, for Task #1, if frame size for the 2 video files are equal, the diagonal is taken under consideration and the time consumption will be less. Otherwise it will be considerably longer.
- For Task #4, we are using the dimensions reduced by PCA.
- Target dimensionality in the computation of PCA & K-means is assumed to be less than original dimensionality of the input data.

Proposed Solution

Architecture Overview

The following diagram identifies how search is being done:



The solution we have developed compares two videos by comparing similarity between their frames. The frames in turn are compared by comparing similarity between cells in the frames. In case of color histogram, each cell has one vector. SIFT and motion vectors can have more than one vector in each cell.

So in order to compute cell level similarity we compare every vector in one cell with every vector in the other cell. This brute force process is computationally expensive but will be give us effective matches i.e. with minimal misses and false hits. After computing similarity/distance between vectors, we use Hungarian a.k.a Munkres' algorithm[1][6] to find the best possible assignments, which vector in one cell matches effectively with that of the other. The sum of the assignment list we get using the algorithm is perceived to be the similarity between two cells. Since complexity of Hungarian algorithm is high $O(n^3)$ we came up with a greedy version of the Hungarian algorithm that just finds the best matches. i.e. for every vector find the closest in terms of distance and highest in terms of similarity without giving importance to the overall similarity between all the vectors. An attempt to use stable marriage algorithm to obtain this matching did not yield results with the desired complexity and was dropped. This is then used to compute frame level similarity using Hungarian algorithm again. This is further integrated to compute similarity between two video sequences. This procedure is explained in detail in the implementation section. Also, we analysed and researched different metrics for the computation of similarity/distance between different features such as Color Histogram, Motion and SIFT vectors and chose the following metrics presented in the table below after comparing the pros and cons of those metrics. Intersection and cosine similarity were found to be suitable for Color Histogram as the pixel resolutions may vary between different videos. Mahalanobis Distance and Intersection similarity* were preferred over other metrics for SIFT as variance among some dimensions (like X and Y values) seemed high compared to others. The metrics and the features we use it for are listed in the table below. We also do not consider frames with no vectors as considering them would lead to misses.

Task Description Table

Sub Task ID	Feature(s)	Metric Used
1a	Color Histogram	Intersection Similarity
1b	Color Histogram	Cosine Similarity
1c	SIFT Vectors	Intersection Similarity*
1d	SIFT Vectors	Mahalanobis Distance
1e	Motion Vectors	L1 Distance
1f	Motion Vectors	Cosine Similarity
1g	Color Histogram, SIFT and Motion Vectors	Intersection Similarity
1h	Color Histogram, SIFT and Motion Vectors	L1 Distance

***Intersection Similarity developed by using the method covered during the QA Session before midterm.*

Intersection Similarity* $sim_{int}(\vec{a},\vec{b}) = \frac{1}{n}\sum_{i=1..n}\frac{\min(a_i,b_i)}{\max(a_i,b_i)}$, $where\ a_i, b_i \neq 0\ \forall i$

Input / Output File Format:

Input Format:

The input file for the Histogram, SIFT and Motion vectors in the following format

$i;j;k;v_1,v_2,v_3 \ldots v_n$ where i: video number, j: frame number, k: cell number, v_p: vector p

The input files are out_file.chst, out_file.sift and out_file.mvect for Color histogram, SIFT vectors and motion vectors respectively

Task # 1:

For data present in input files, task 1 determines the distance measure between two video file and displays them to the user. Refer to the interface specification of task 1 and its subtasks or further details of parameters, output format. README file also provides a brief overview of the input/output format.

Task # 2:

For data present in input files, task 2 retrieves and displays k most similar subsequence of video frames based on input provided to the user. Refer to the interface specification of task 2 and its subtasks for further details of parameters, output format. README file also provides a brief overview of the input/output format.

Task # 3:

There are two output files:

1. Containing the data in the reduced dimension
2. Containing the reduced dimension info in terms of <original index, score>

The first file is of the following format:

The rows represent the data and the columns represent the dimension of the data

$i;j;k;v'_1,v'_2,v'_3 \ldots v'_n$ where i: video number, j: frame number, k: cell number, v'_p: vector p in reduced dimension

The second file is of the following format:

Each row represent the d – dimensions that is reduced

The file is in the following format and has d lines for each dimension

The first line is for the first dimension, in PCA it represents the first principal component

(original_index(m),$score_1$), (original_index(n),$score_2$),…., (original_index(o),$score_n$)

Where $score_1 >= score_2 >= \ldots. >= score_n$ is in non-increasing order

Task # 4:

For data present in input files, task 4 retrieves and displays k most similar subsequence of video frames based on input provided to the user for the reduced dimension data from task 3. Refer to the interface specification of task 4 for further details of parameters, output format. README file also provides a brief overview of the input/output format.

Implementation

Motion Vectors pruning

Motion vectors extracted from Phase 1 of the project have the source and the destination co-ordinates in them which give the direction in which a macroblock is moving. Since our focus is on moving objects in the videos and not compression, motion vectors with the same source and destination (i.e, destination-source = (0,0)) are not of much use to us.

As these static redundant motion vectors could also lead us to False hits, we remove such vectors from the original out_file.mvect. Furthermore, to improve effectiveness, we append difference between the destination and source instead of the source and write it to out_file.mvect.filtered file. This file is used for motion vector comparisons for all the given tasks (1 to 4).

Format of the motion vector is as follows:

Video Number; Frame Number; Cell Number; Source, Width of the macroblock, Height of the macroblock, Absolute Detination Position(x) , Absolute Destination Position(y), Difference between the Absolute Destination and Source Positions (x), Difference between the Destination and Source Positions(y)

Source indicates if the block came from the past or future. -1: if past , +1 if future.

Task 1

1. The program task1.py takes 3 command line arguments as input
    - Sub Task ID (1a-1h)
    - The video number ranging from 0 to number of videos provided in the directory given (i.e., 48). Represented as v_i
    - The other video with which v_i is going to be compared with v_j
2. Decide which feature and metric to use based on the Sub Task ID
3. The features are extracted from the input files we got from phase 1.
4. Next compare the two given videos (from frame 0 till the end of each)
    a. Compute similarity between frames of the videos. Ignore certain comparisons that would be meaningless (e.g. Comparing the first frame of one video with the last one or vice-versa, while taking into consideration number of frames in both the videos. The idea is to maintain the frame chronology between 2 videos.
    b. If n_i is the number of frames in video v_i and n_j is that of the other, do not compare the k^{th} frame of v_i with the frame n_j if $j>n_j-k+1$ and vice versa.
    c. For each frame F_k and F_l of the two videos given, compute their similarity by computing similarity between cells of each frames.
    d. For R, the resolution from phase 1 (=2 in our case), all R^2 cells of frames of F_k and F_l are compared with each other.
    e. Each cell is in turn compared by comparing all each of the vectors in one cell to the other using the Metric type passed as input.

f. If there are ‘n’ vectors in one cell and ‘m’ in the cell, all of them are compared and the values are stored in a n by m matrix. (If the metric used was a distance measure, then it is converted to similarity using the expression “Similarity=1/(1+Distance)” for computational convenience. The values are negated too for a similar reason.
g. The Hungarian algorithm also known as the Munkres[1][6] algorithm is then run on this matrix to find the best possible assignment. (i.e.) for a vector vr_i (i<=n) in cell C_A and a vector vr_j (j<=m) of cell C_B, find assignments (vr_i,vr_j) i.e. which is basically each value in the n X m matrix such that the total sum of such assignments is minimum.
h. Since the complexity of the above algorithm is high ($O(n^3)$). We came up with a greedy brute force method of finding assignments that runs in $O(n^2)$ whenever n or m is high to improve efficiency.
i. Now the mean (divide by max(n,m) for the steps mentioned below as well) of minimal sum of the assignments obtained above is the similarity between two cells.
j. Similarity between all cells is computed and an assignment is done using Hungarian algorithm again using step i. The absolute value of the mean is the similarity between two frames F_k and F_l.
k. After computing similarity between all frames, a dynamic programming approach is applied to find the best possible match between the two videos v_i and v_j
l. This is done by finding a path that maximizes the sum (i.e. similarity). By backtracking, we can also find the frame sequences that lead to the highest value.
m. The mean of this value is computed. If the metric used was a distance metric, the similarity value is converted to distance using the equation, “Distance=(1/Similarity)-1” (s.t Similarity>0)
n. This value is the final video similarity between two videos.

5. If overall similarity was to be computed using Color Histogram, SIFT and Motion vectors, similarity between the videos is computed using the above steps and the values are converged into a single value.

Task 2

1. The program task2.py takes 6 command line arguments as input
   - Sub Task ID (1a-1h), indicating one of the eight methods used in task 1.
   - The video number ranging from 0 to number of videos provided in the directory given (i.e., 48). Represented as v_i
   - The frame number ‘a’ that corresponds to the start of the video subsequence from the video v_i that is to be matched with all the videos in the directory
   - The frame number ‘b’ that corresponds to the end of the video subsequence from the video v_i that is to be matched with all the videos in the directory
   - The path to the directory containing all the videos given
   - The value ‘k’, which indicates the top ‘k’ most subsequences be retrieved.
2. Decide which feature and metric to use based on the Sub Task ID
3. The features are extracted from the input files we got from phase 1.
4. From the features, select only vectors in the frame range [a,b] of video v_i and assume that to an independent video itself (say Temp).
5. Now compare this Temp video with all the videos in the given directory by following the steps mentioned under parts 4 (and 5 if required as per the input given) of Task 1.
6. As mentioned in the ‘4l’ section of Task 1, it is possible to back track in the matrix used to find the path with maximum similarity. For this task, we use the column numbers that indicates the beginning (C1) and end (C2) of such a path. We calculate the mean by dividing the overall value by the (C2-C1). Therefore, C1 to C2 indicate the best subsequence from a video and mean indicates the similarity/Distance.

7. After calculating the similarity for all videos in the directory, the frame sub sequences (containing the video V and frame range [x,y]) are sorted in the order of decreasing similarity (or increasing distance)
8. The top 'k' of those sub sequences are selected and played (displayed in the form of a video) after playing the original query (i.e. v_i [a,b])

TASK #3

Task 3 (a-c)

1. Read the input data as n x k matrix where n is the number of data and k is the number of original dimensions
2. Compute the co-variance matrix for the input matrix (k x k)
3. Do Eigen Decomposition for the co-variance matrix matrix
4. Sort the eigen values and the corresponding eigen vectors in non-increasing order
5. Select the first k eigen vectors
6. Find the dot product of the data matrix and the eigen vectors
7. This gives the transformed data matrix in the new vector space
8. After this select the top k eigen vectors and print it in the file based on the corresponding dimension index and score

Task 3 (d-f)

1. Read the new number of dimensions d from the user
2. Read the input data as n x k matrix where n is the number of vectors and k is the number of original dimensions
3. Perform K-Means clustering on the matrix with d as the number of clusters
4. Store the cluster centers into an array
5. For every vector stored in the input matrix, perform the following
6. Compute the dot product of the vector with each of the d cluster centers.
7. Now, these d values form the new vector in the reduced dimension space.
8. Write the output n x d matrix into the file

TASK #4

Task 4 is precisely the same as task 2 except that the step 3 for task 4 is to extract features from the files that are the output from Task 3. i.e. It works on a reduced dimensional space.
For this task we assumed the reduced dimensional space mentioned in the problem statement corresponds to PCA and the code runs based on that. (We can perform the same task with the reduced dimensions from k-means by modifying the code a bit.)

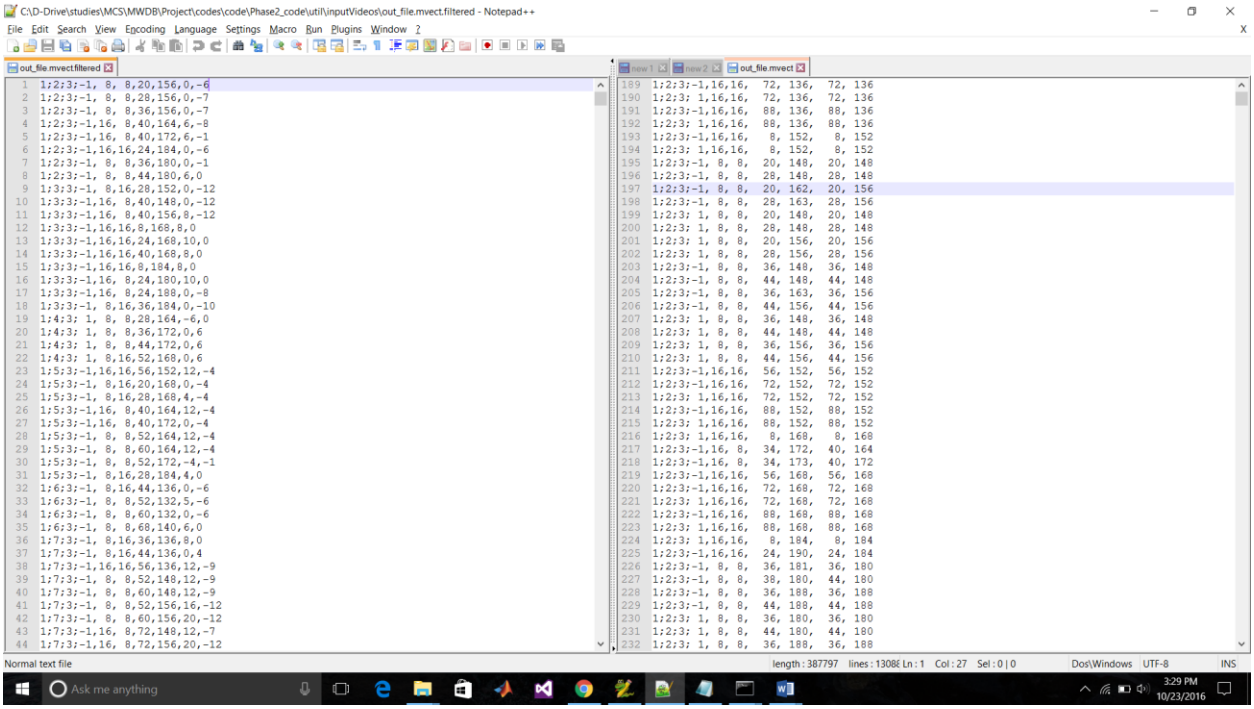
Interface Specification

Motion Vector Pruning

The script processMvect.py parses the original out_file.mvect and filters motion vectors that have the same source and destination.

Usage: python processMvect.py

The script is internally invoked by the each of the tasks as and when the motion vectors are needed.



The scripts writes the pruned motion vectors to out_file.mvect.filtered file in the form:

<Video No, Frame No, Cell No, Source, Macroblock width, Macroblock height, Destination (x), Destination (y), Destination(x)-Source(x), Destination(y)-Source(y)>

(The original motion vectors in out_file.mvect are of the form:

<Video No, Frame No, Cell No, Source, Macroblock width, Macroblock height, Source(x), Source(y), Destination (x), Destination (y)>)

Where Source(x) and Destination(x) represent the X co-ordinate of the source and destination respectively. Similarly Source(y) and Destination(y) represent the Y co-ordinate of the source and destination respectively.

Task #1:

There are 8 subtasks under task 1: To execute these we must run task1.py with 3 more command line arguments. They are the subtask ID viz., 1a,1b,1c..1h etc., each corresponding to one of the 3 features Color Histogram, SIFT and Motion Vectors and different distance and similarity metrics.

Task 1a

Usage: python task1.py 1a v_i v_j

The program displays the Intersection similarity between 2 videos using Color Histogram and the time taken for execution. The screenshot shows the Intersection Similarity between two videos 0 and 3 (i.e. 6x_SQ_BL_TM_BR_Check.mp4 and 6x_SQ_BL_TR_Check.mp4)

```
Command Line : 1a 0 3
Intersection Similarity=0.974470677823
Time Taken for execution:0.70360264687 seconds
```

Task 1b

Usage: python task1.py 1b v_i v_j

The program displays the Cosine similarity between 2 videos using Color Histogram and the time taken for execution. The screenshot shows the Cosine Similarity between two videos 0 and 3 (i.e. 6x_SQ_BL_TM_BR_Check.mp4 and 6x_SQ_BL_TR_Check.mp4)

```
Command Line : 1b 0 3
Cosine Similarity=0.999163229079
Time Taken for execution:0.713040090897 seconds
```

Task 1c

Usage: python task1.py 1c v_i v_j

The program displays the Intersection Similarity (Method 2) between 2 videos using SIFT vectors and the time taken for execution. The screenshot shows the Similarity between 2 videos 0 and 12(6x_SQ_BL_TM_BR_Check.mp4 and 6x_TR_BL_TR_Check.mp4), and the time taken for execution.

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task1.py 1c 0 12
Intersection Similarity M2=0.859796559813
Time Taken for execution:1102.96487892 seconds
```

Task 1d

Usage: python task1.py 1d v_i v_j

The program displays the Mahalanobis Distance between 2 videos using SIFT vectors and the time taken for execution. The screenshot shows the Mahalanobis Distance between 2 videos 2 and 5(6x_SQ_BL_TM_BR_White.mp4 and 6x_SQ_BL_TR_White.mp4), and the time taken for execution.

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task1.py 1d 2 5
Mahalanobis Distance=0.570894212956
Time Taken for execution:199.941101628 seconds
```

Task 1e

Usage: python task1.py 1e v_i v_j

The program displays the Motion Vector similarity between the two given videos using L1 Distance and the time taken for execution. The screenshot shows the L1 Distance between videos 0 and 0 (i.e. 6x_SQ_BL_TM_BR_Check.mp4 and 6x_SQ_BL_TM_BR_Check.mp4)

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task1.py 1e 0 0
L1 Distance=0.0
Time Taken for execution:1.65893361448 seconds
```

Task 1f

Usage: python task1.py 1f v_i v_j

The program displays the Motion Vector similarity between the two given videos using Cosine Similarity and the time taken for execution. The screenshot shows the Cosine Similarity between videos 0 and 3 (i.e. 6x_SQ_BL_TM_BR_Check.mp4 and 6x_SQ_BL_TR_Check.mp4)

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task1.py 1f 0 3
Cosine Similarity=0.792992201051
Time Taken for execution:1.63590384465 seconds
```

Task 1g

Usage: python task1.py 1g v_i v_j

The program displays the Overall Similarity (Color Histogram, SIFT and Motion Vector) using Intersection Similarity and the time taken for execution. The screenshot shows the same between videos 2 and 14 (i.e. 6x_SQ_BL_TM_BR_White.mp4 and 6x_TR_BL_TR_White.mp4)

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task1.py 1g 2 14
Intersection Similarity=0.321808913755
Time Taken for execution:36.4052114796 seconds
```

Task 1h

Usage: python task1.py 1h v_i v_j

The program displays the Overall Similarity (Color Histogram, SIFT and Motion Vector) using L1 Distance and the time taken for execution. The screenshot shows the same between videos 1 and 13 (i.e. 6x_SQ_BL_TM_BR_Noise.mp4 and 6x_TR_BL_TR_Noise.mp4)

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task1.py 1h 1 13
L1 Distance=17.7998863932
Time Taken for execution:44.3433733732 seconds
```

Task #2:

Usage: python task2.py <One of the 8 methods from task 1> v_i a b <Path to the directory containing videos> k

The program displays the Query video and the frame range given [a,b] and top 'k' most similar subsequences of the form:

<the video name> <Start frame> <End Frame>

The program also displays the query and the results in the form of a video.

E.g.: python task2.py 1a 0 11 21 C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\ 16

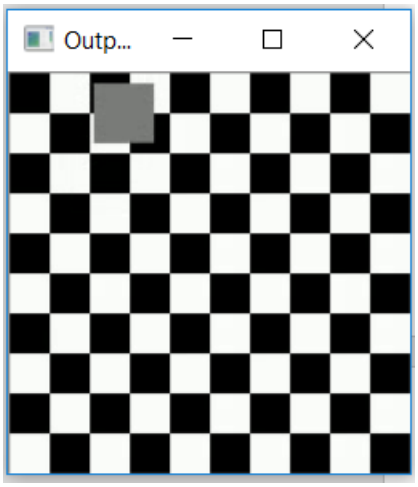
This command retrieves the top 16 most similar video sequences from the frames [11,21] of video 0 (6x_SQ_BL_TM_BR_Check.mp4) using Color Histogram Intersection Similarity.

It also prints the time taken for execution.

The screenshot below displays the results in text format and the videos play alongside.

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task2.py 1a 0 11 21 C:\\D-Drive\\studies\\MCS\\MW
DB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\ 16
Query: C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TM_BR_Check.mp4 11 21


Top 16 most similar sequences are:
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TM_BR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_75_25_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_25_75_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_BL_TM_BR_Check.mp4 11 22
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_50_50_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_25_75_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_TL_BR_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_50_50_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_BL_TR_Check.mp4 11 24
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_TL_BR_Check.mp4 11 24
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_75_25_Check.mp4 11 26
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TR_Check.mp4 11 27
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TR_Check.mp4 11 29
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_TL_BR_Check.mp4 11 29
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_TL_BR_Check.mp4 11 29
Time Taken for execution:113.21769396 seconds
```

The below one is the snapshots of one such video sequences.



Task #3:

Task 3(a)

Usage: python pcaHist.py

The program reduces the dimensions of the given Histogram data to ‘d’ given dimension using PCA which is provided as input value when the program prompts. The program reduces the dimension of the data and stores the new data in a file, the program also writes the dimensions with the <original_index,score> format.

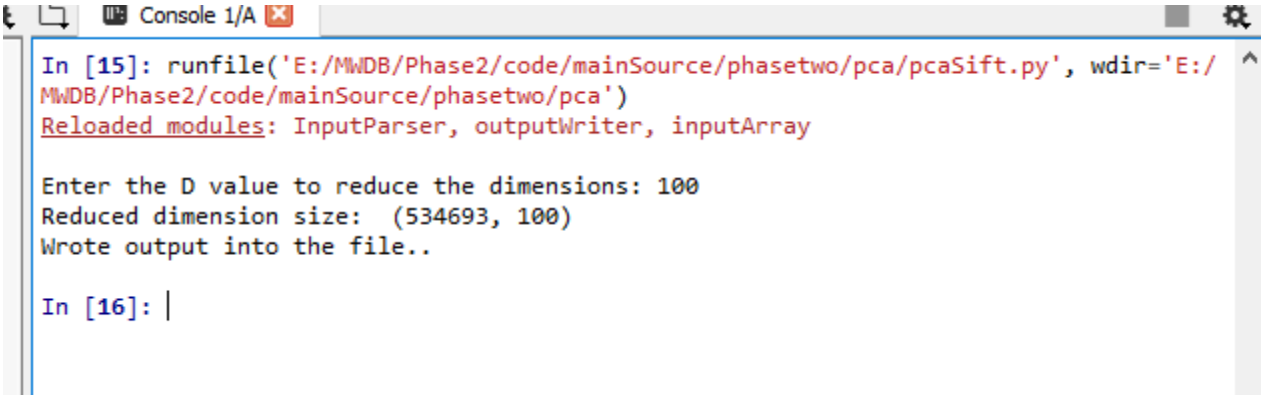
```
In [14]: runfile('E:/MWDB/Phase2/code/mainSource/phasetwo/pca/pcaHist.py', wdir='E
MWDB/Phase2/code/mainSource/phasetwo/pca')
Reloaded modules: InputParser, outputWriter, inputArray

Enter the D value to reduce the dimensions: 10
Reduced dimension size:  (7872, 10)
Wrote output into the file..

In [15]:
```

new 1 | new 2 | out_file.mvect | out_file_d.cpca

```
1;1;1;869.647992845,-5026.09936308,-1028.65623369,-159.84223864,837.441544612,-250.896609051,-89.7845439695,-76.0390554735,37.0548271347,-5.08233539798
1;1;2;1479.21966874,-5142.66261229,-1685.48561421,14.4373800201,-190.065401502,-97.7120961149,48.3779415928,31.7543278162,29.3430396764,5.11272090683
1;1;3;1542.19701563,-4628.4503411,-1364.016048,-662.770968541,-390.032659663,-55.7722688907,21.8556872534,-5.23246543247,-25.4885888556,65.3998645687
1;1;4;518.101753649,-5432.56292247,-2116.10474761,18.8600145072,-271.200351468,-75.2784844604,19.6211444955,9.72400467831,31.887587032,4.57150824724
1;2;1;869.647992845,-5026.09936308,-1028.65623369,-159.84223864,837.441544612,-250.896609051,-89.7845439695,-76.0390554735,37.0548271347,-5.08233539798
1;2;2;1479.21966874,-5142.66261229,-1685.48561421,14.4373800201,-190.065401502,-97.7120961149,48.3779415928,31.7543278162,29.3430396764,5.11272090683
1;2;3;1624.41996529,-4519.95855326,-1284.86159828,-720.705506602,-358.132198369,42.3146811126,-23.5898621816,20.5077663448,-27.4293585608,9.3174413678
1;2;4;518.101753649,-5432.56292247,-2116.10474761,18.8600145072,-271.200351468,-75.2784844604,19.6211444955,9.72400467831,31.887587032,4.57150824724
1;3;1;869.647992845,-5026.09936308,-1028.65623369,-159.84223864,837.441544612,-250.896609051,-89.7845439695,-76.0390554735,37.0548271347,-5.08233539798
1;3;2;1479.21966874,-5142.66261229,-1685.48561421,14.4373800201,-190.065401502,-97.7120961149,48.3779415928,31.7543278162,29.3430396764,5.11272090683
1;3;3;1661.67827892,-4528.96538362,-1230.63178837,-665.997704201,-328.390825963,20.8407123038,12.6031813067,1.71882436316,-31.0199105569,4.41075551215
1;3;4;518.101753649,-5432.56292247,-2116.10474761,18.8600145072,-271.200351468,-75.2784844604,19.6211444955,9.72400467831,31.887587032,4.57150824724
1;4;1;869.647992845,-5026.09936308,-1028.65623369,-159.84223864,837.441544612,-250.896609051,-89.7845439695,-76.0390554735,37.0548271347,-5.08233539798
1;4;2;1479.21966874,-5142.66261229,-1685.48561421,14.4373800201,-190.065401502,-97.7120961149,48.3779415928,31.7543278162,29.3430396764,5.11272090683
1;4;3;1728.77955905,-4302.36060015,-1094.77270159,-613.690449963,-222.047361344,162.995656364,12.1841190704,-39.99814329,-74.8385583702,-52.0254506088
1;4;4;518.101753649,-5432.56292247,-2116.10474761,18.8600145072,-271.200351468,-75.2784844604,19.6211444955,9.72400467831,31.887587032,4.57150824724
1;5;1;869.647992845,-5026.09936308,-1028.65623369,-159.84223864,837.441544612,-250.896609051,-89.7845439695,-76.0390554735,37.0548271347,-5.08233539798
1;5;2;1479.21966874,-5142.66261229,-1685.48561421,14.4373800201,-190.065401502,-97.7120961149,48.3779415928,31.7543278162,29.3430396764,5.11272090683
1;5;3;1672.22913034,-4457.11787407,-964.68507267,-700.801090039,-159.299059173,53.8299931447,48.1301024722,-20.2556181144,-87.0677684751,-64.6363929764
1;5;4;518.101753649,-5432.56292247,-2116.10474761,18.8600145072,-271.200351468,-75.2784844604,19.6211444955,9.72400467831,31.887587032,4.57150824724
1;6;1;869.647992845,-5026.09936308,-1028.65623369,-159.84223864,837.441544612,-250.896609051,-89.7845439695,-76.0390554735,37.0548271347,-5.08233539798
1;6;2;1479.21966874,-5142.66261229,-1685.48561421,14.4373800201,-190.065401502,-97.7120961149,48.3779415928,31.7543278162,29.3430396764,5.11272090683
```

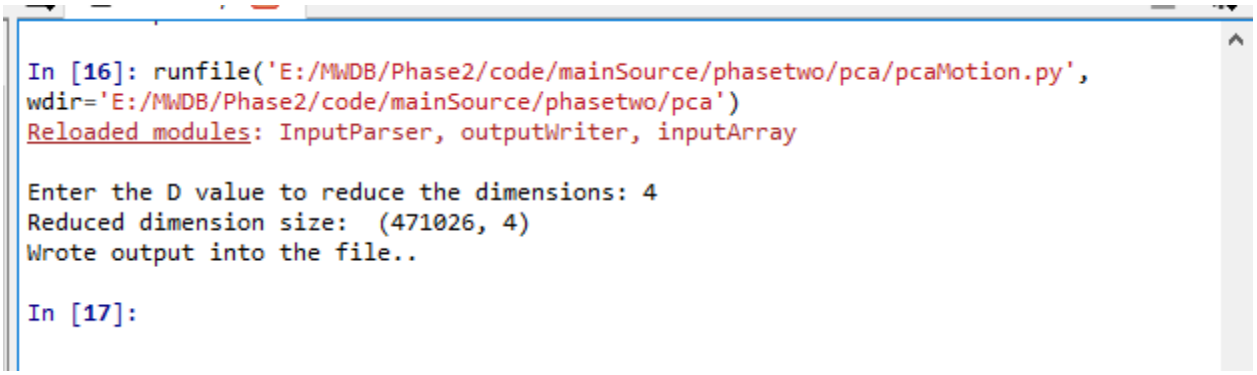
Task 3(b)

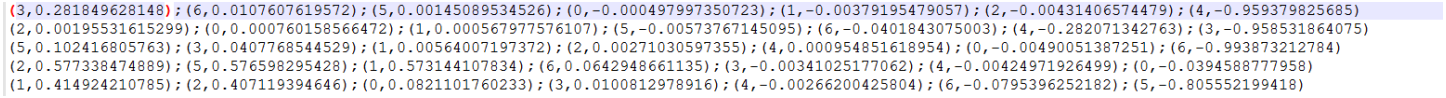
The program reduces the dimensions of the given Sift data to ‘d’ given dimension using PCA which is provided as input value when the program prompts. The program reduces the dimension of the data and stores the new data in a file, the program also writes the dimensions with the <original_index,score> format.



Task 3(c)

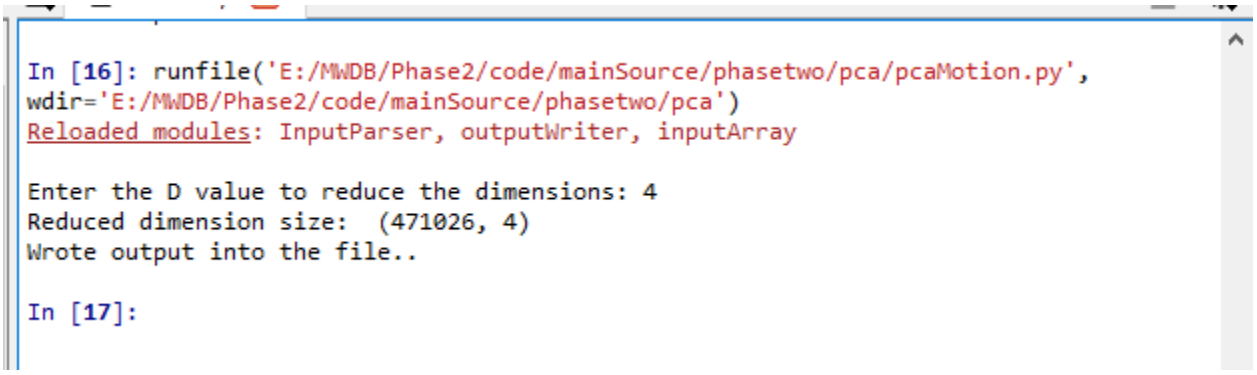
The program reduces the dimensions of the given Motion vector data to ‘d’ given dimension using PCA which is provided as input value when the program prompts. The program reduces the dimension of the data and stores the new data in a file, the program also writes the dimensions with the <original_index,score> format.





Task 3(d)

The program reduces the dimensions of the given Histogram vector data to ‘d’ given clusters using k-Means which is provided as input value when the program prompts. The program clusters the data and stores the new data in a file, the program also writes the clusters with the <original_index,score> format.



Task 3(e)

The program reduces the dimensions of the given SIFT vector data to ‘d’ given clusters using k-Means which is provided as input value when the program prompts. The program clusters the data and stores the new data in a file, the program also writes the clusters with the <original_index,score> format.

```
In [9]: runfile('E:/MWDB/Phase2/code/mainSource/phasetwo/pca/kmeansSift.py',
wdir='E:/MWDB/Phase2/code/mainSource/phasetwo/pca')
Reloaded modules: InputParser, outputWriter, inputArray
Enter the number of dimensions

10
Output vector generated of dimension:
534693 10

In [10]:
```

Task 3(f)

The program reduces the dimensions of the given motion vector data to ‘d’ given clusters using k-Means which is provided as input value when the program prompts. The program clusters the data and stores the new data in a file, the program also writes the clusters with the <original_index,score> format.

```
In [12]: runfile('E:/MWDB/Phase2/code/mainSource/phasetwo/pca/kmeansMotion.py',
wdir='E:/MWDB/Phase2/code/mainSource/phasetwo/pca')
Enter the number of dimensions

4
Output vector generated of dimension:
471026 4

In [13]:
```

```
(3,131.764155358);(4,61.7800655124);(1,12.1403837155);(2,12.0393074403);(5,3.21619092185);(0,-0.664014974263);(6,-0.944314459523)
(4,153.775166471);(3,42.6102624364);(1,12.4778691735);(2,12.4245985116);(5,2.82256169213);(0,-0.567567567568);(6,-3.78613396005)
(3,67.283887468);(4,54.6155157715);(1,12.1329923274);(2,12.0716112532);(5,2.80392156863);(6,0.576300085251);(0,-0.577152600171)
(3,155.482728078);(4,152.302922941);(1,12.5066430469);(2,12.2621789194);(6,3.90921169176);(5,3.15234720992);(0,-0.628875110717)
(4,108.7730029);(3,104.108621144);(1,11.9441075666);(2,11.7732665436);(5,3.01001845505);(6,0.258634326391);(0,-0.679936725547)
```

Task #4 :

Usage: python task4.py <One of the 8 methods from task 1> v_i a b <Path to the directory containing videos> k

The program displays the Query video and the frame range given [a,b] and top ‘k’ most similar subsequences of the form:

<the video name> <Start frame> <End Frame>

The program also displays the query and the results in the form of a video.

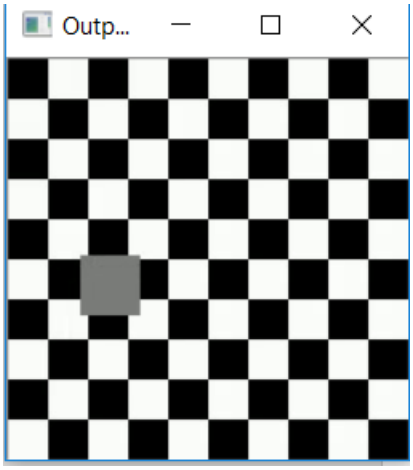
E.g.: python task4.py 1a 0 11 21 C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\ 16

This command retrieves the top 16 most similar video sequences from the frames [11,21] of video 0 (6x_SQ_BL_TM_BR_Check.mp4) using Color Histogram Intersection Similarity in the reduced dimension space (using PCA and d=10, results obtained from task3)

It also prints the time taken for execution.
The screenshot below displays the results in text format and the videos play alongside.

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task4.py 1a 0 11 21 C:\\D-Drive\\studies\\MCS\\MW
DB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\ 16
Query: C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TM_BR_Check.mp4 11 21


Top 16 most similar sequences are:
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TM_BR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_75_25_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TR_Check.mp4 11 22
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_BL_TM_BR_Check.mp4 11 22
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_25_75_Check.mp4 11 22
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_50_50_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_25_75_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_50_50_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_TL_BR_Check.mp4 11 23
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_BL_TR_Check.mp4 11 24
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_TL_BR_Check.mp4 11 24
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_75_25_Check.mp4 11 26
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_TL_BR_Check.mp4 11 29
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_TL_BR_Check.mp4 11 33
Time Taken for execution:41.4719246814 seconds
```

The below one is the snapshots of one such video sequences.



Task 4-h, Overall Similarity in the reduced dimension space (L1 Distance)

```
C:\D-Drive\studies\MCS\MWDB\Project\codes\code\Phase2_code\util>python task4.py 1h 0 11 21 C:\\D-Drive\\studies\\MCS\\MW
DB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\ 16
Query: C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TM_BR_Check.mp4 11 21


Top 16 most similar sequences are:
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TM_BR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_BL_TR_Check.mp4 11 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_BL_TM_BR_Check.mp4 10 20
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_BL_TR_Check.mp4 10 21
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TR_Check.mp4 23 38
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TR_Check.mp4 23 46
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_75_25_Check.mp4 21 45
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_75_25_Check.mp4 19 43
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_50_50_Check.mp4 32 44
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_25_75_Check.mp4 31 48
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_BL_TM_BR_25_75_Check.mp4 23 42
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_SQ_TL_BR_Check.mp4 9 19
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_BL_TM_BR_50_50_Check.mp4 25 44
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\TR_TL_BR_Check.mp4 20 35
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\SQ_TL_BR_Check.mp4 19 38
C:\\D-Drive\\studies\\MCS\\MWDB\\Project\\codes\\code\\Phase2_code\\P2DemoVideos\\6x_TR_TL_BR_Check.mp4 8 19
Time Taken for execution:10554.613436 seconds
```

System Requirements

Operating System: Windows
RAM: 8 GB
Software tools : Python, PyScripter, WinPython tool, Spyder
Libraries: Numpy 1.8, scikit-learn, Munkres[1][6], Scipy, OpenCV[2] python extension
Additional tools: MATLAB, MS Visual studio

Related Work

Even though most of the distance measure considered here are Euclidean, there are models which improve upon these distance measures by considering other measures for determining similarity between videos for effective content-based search. 'Efficient Video Similarity Measurement With Video Signature'[21] considers similarity between videos by generating video signatures for frames in given video to uniquely identify and compare them. 'Beyond Distance Measurement: Constructing Neighbourhood Similarity for Video Annotation'[20] on the other hand introduces a novel approach called neighbourhood similarity which considers local and sample label distribution for given videos and identifies this approach to be superior than Euclidean measures used in traditional systems. 'Kullback-Leibler similarity measures for effective content based video retrieval'[19] uses K-L distance to determine the best fit match for videos. Different approaches can be used to determine similarity based on the intended use of the application and the consideration for user subjectivity.

Conclusion

The proliferation of video content on the Web makes similarity detection an indispensable tool in Web data management, searching, and navigation. Video similarity determination is a complex procedure and if content-based retrieval is added to it then it adds the additional overhead of user subjectivity to determine similarity for videos. As the concern here is to determine video subsequence similarity for a given set of frames consideration for subjectivity is removed but the similarity determination based on color histogram, SIFT and motion vectors is still a challenge as the distance measure to determine similarity must be in correspondence to visual perception. Identifying and determining appropriate distance measures was a challenge as the same measures need to yield correct result even after dimensionality reduction using PCA and k-means as there is a possibility of overfitting and underfitting. Much more complex systems[19][20][21] exist that determine similarity based on a multitude of different considerations like neighbourhood similarity, Kullback-Lieber distance and video signature and could potentially yield better results for content-based video retrieval. We also realized the scale of efficiency dimensionality reduction can lead us to. This is clearly seen from the outputs of task 2 and 4.

Bibliography

[1] Bourgeois, François and Jean-Claude Lassalle (1971)"An extension of the Munkres algorithm for the assignment problem to rectangular matrices" Communications of the ACM 14.12 (1971): 802-804.

[2] OpenCV (2016). Accessed 11 Oct. 2016. URL: http://docs.opencv.org/3.0/beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html

[3] Wikipedia (K-means Clustering) (2016), Accessed 11 Oct. 2016. URL: https://en.wikipedia.org/wiki/K-means_clustering

[4] Wikipedia (Hungarian Algorithm) (2016). Accessed 11 Oct. 2016. URL: https://en.wikipedia.org/wiki/Hungarian_algorithm

[5] Wikipedia (Stable Marriage Problem) (2016). Accessed 13 Oct. 2016. URL: https://en.wikipedia.org/wiki/Stable_marriage_problem

[6] Python Community (Munkres) (1990-2016). Accessed 13 Oct. 2016. URL: https://pypi.python.org/pypi/munkres/

[7] Wikipedia (Decorator Pattern) (2016). Accessed 14 Oct. 2016. URL: https://en.wikipedia.org/wiki/Decorator_pattern

[8] Simeon Franklin (2012). Accessed 15 Oct 2016. URL: http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/

[9] Numpy ref.(Eigen decomp., Covariance matrix, Dot prod.) (2016). Accessed 16 Oct 2016. URL: https://docs.scipy.org/doc/numpy/reference/generated/

[10] Scikit learn (2010-2016), Accessed. 17 Oct 2016. URL: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[11] Wikipedia (Jaccard Index) (2016), Accessed 22 Oct 2016. URL: https://en.wikipedia.org/wiki/Jaccard_index

[12] Wikipedia (Cosine Similarity) (2016), Accessed 22 Oct 2016. URL: https://en.wikipedia.org/wiki/Cosine_similarity

[13] Wikipedia (Mahalonobis Distance) (2016), Accessed 22 Oct 2016. URL: https://en.wikipedia.org/wiki/Mahalonobis_distance

[14] Wikipedia (Taxicab geometry) (2016), Accessed 22 Oct. 2016. URL: https://en.wikipedia.org/wiki/Taxicab_geometry

[15] Wikipedia (Principal Component Analysis) (2016), Accessed 22 Oct. 2016. URL: https://en.wikipedia.org/wiki/Principal_component_analysis
[16] Chesti Altaff Hussain, Dr. D. Venkata Rao, T. Praveen 'Color Histogram Based Image Retrieval'. In International Journal of Advanced Engineering Technology IV/III/July-Sept.,2013/63-66, Page 1

[17] David G. Lowe 'Distinctive Image Features from Scale-Invariant Keypoints', January 5, 2004, Page 2

[18] Aroh Barjatya ' Block Matching Algorithms For Motion Estimation', DIP 6620 Spring 2004, Page 1

[19] Priya R, Shanmugam T N, Bhaskaran R 'Kullback-Leibler similarity measures for effective content based video retrieval'. Imaging Science Journal. Sep2013, Vol. 61 Issue 7, p541-555 Page 1-2

[20] Meng Wang, Xian-Sheng Hua, Jinhui Tang Richang Hong 'Beyond Distance Measurement: Constructing Neighborhood Similarity for Video Annotation' IEEE Transactions On Multimedia, Vol. 11, No. 3, April 2009 Page 1-2

[21] Sen-ching,Samson Cheung and Avideh Zakhor,'Efficient Video Similarity Measurement With Video Signature', IEEE Transactions On Circuits And Systems For Video Technology, Vol. 13, No. 1, January 2003, Page 1-2

Appendix

UTHARA KEERTHI
Computed Distance measures and implemented Dimensionality reductions using K-means for Color Histogram, SIFT and Motion Vectors.
DARSHAN SHETTY
Developed a strategy to process input files from Phase #1 and generate output video files, in Phase #2.
BHARATH KUMAR SURESH
Designed and implemented the architecture and integrated different modules like InputParser / distance measure etc., in Phase #2.
DHANANJAYAN SANTHANAKRISHNAN
Worked on PCA code implementation for Histogram, Sift and Motion Vectors.
SHIVAM GAUR
Comprehensive implementation of PCA and K-means and Report creation for Phase #2.