Video Analysis and Feature reduction

# Abstract

The recent technology growth in database has facilitated numerous luminaries to represent data in non-redundant form. The data extraction and processing has grown beyond just relational database, unstructured database to multimedia database. The vast amount of data delivered from a video stream provides the need to identify the contents in a video. A higher level of content abstraction and investigation in the distinct region is required to identify the video content. A video can contain several points of interest in every frame. We extract the features such as color, key points and motion vectors and use the extracted “feature description” to find video similarities and do subsequence search using various distance similarity measures. Also, since the multimedia objects are of huge size, we focus on reducing the dimensionality of the individual data sets pertaining to color, sift key points and motion vector using PCA (Principal component analysis and k-means) and do similarity and subsequence search in the reduced data. We also present results of the subsequence search in the database containing similar videos and visually represent the most similar videos based on the match percentage obtained from distance-similarity functions.

Group Members:

Anoop Jatavallabha Vijayakumar

Deepak Soundararajan

Narendra Kumar Sampath Kumar

Ravikiran Tangirala

Santosh Mandya Jayaram

Keywords:

Video analysis, Histogram, Principal Component Analysis, K-means, Motion vectors, SIFT vectors, Histogram.

Introduction:

Data redundancy is one of the biggest challenge in Multimedia data management. When the input data to an algorithm is too large to be processed and it is suspected to be redundant, it directly has an impact on the reliability of the product. To overcome the data redundancy, the input data can be transformed into reduced set of feature vectors (process called as Feature Selection). Feature Selection largely relies on the purpose of the application. For example, sports video summarization usually focuses more on the motion of foreground athletes and balls, while generic-purpose applications may concern more about the background scenes. This phase of the project focuses on utilizing the extracted color, SIFT and motion vectors from the previous phase and finds the similarity between the given two videos using similarity measures such as (Intersection similarity, Tanimoto similarity coefficient, Manhattan distance, Euclidean distance) and find the nearest k similar videos in the database in the nonreduced and reduced database obtained through PCA and kmeans.

Terminology:

*Feature:*

A Feature is a piece of information which is relevant for solving the computational task related to a certain application.

*Feature vectors:*

When two or more different features are extracted, resulting in two or more feature descriptors at each image point. The organized collection of the information provided by two or more feature descriptors as the elements of one single vector is called Feature vector

*Feature Detection:*

Feature detection refers to methods that aim at computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not

*Feature Extraction:*

Feature Extraction is a process of extracting features from every frame in a video, independently of past and future frames.

*Histogram:*

A histogram is a graphical representation of the distribution of numerical data. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent, and are usually equal size. For digital images, a colour histogram represents the number of pixels that have colours in each of a fixed list of colour ranges. In this project we deal with a gray scale image. So a histogram for gray scale image is the intensity of each pixel. The intensity range is between (0,255)

*Motion Vectors:*

A motion vector is the key element in the motion estimation process. It is used to represent a macroblock in a picture based on the position of this macroblock (or a similar one) in another picture, called the reference picture.

*SIFT (Scale Invariant-feature transform):*

Scale-invariant feature transform (or SIFT) is an algorithm in computer vision to detect and describe local features in images

*Frames:*

A frame is an electronically coded still image in video.

*Euclidean distance:*

This is the straight-line distance between two points*.* Lower the value of Euclidian distance higher will be the similarity between the vectors.

Euclidian distance for n dimensions is defined as:  
deuclidian (p, q) = ((p1 - q1)2+( p2 - q 2)2+( p3 - q3)2+( p4 - q4).+. . . . . . +( pn - qn)2)1/2

*Manhattan distance:*

Manhattan distance is measured as the distance between two points measured along axes at right angles. Similar to Euclidian distance, lower the value of Manhattan distance higher will be the similarity between the vectors.  
dmanhattan (p, q) =∑\_(i=1)^n|pi - qi|  
Where p, q are vectors.

*Intersection Similarity:*

It measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. Higher the value of the similarity function higher will be the similarity between the sample sets.

*Tanimoto Similarity:*

The Tanimoto score can be used to compare vectors. You can compare any two vectors, ti and tj, using Tanimoto by computing maximum( ti,tj )-minimum( ti,tj ) /maximum( ti,tj ). Which would give a number between 0 and 1 representing how "similar" they are [11].

Goal Description:

* Find the video similarity between two files based on the in\_file.chst, in\_file.sift, in\_file.mvect created for a given set of video files and parameter settings.
  + - Color similarity between two video files (Intersection similarity, Tanimoto similarity)
    - SIFT similarity between two video files (Manhattan distance, Euclidean distance)
    - Motion similarity between two video files (Intersection similarity of motion vector’s displacement and angle radian, Manhattan distance of the motion vector’s angle radian)
* Given a video file vi and frame range [a,b] , an integer k and with the choice of distance/similarity methods, the goal is to find k - similar videos with the given frame range.
* Given a dimension d, the goal is to reduce the input database in\_file.chst, in\_file.sift, in\_file.mvect to the reduced dimensional representation in new vector space using PCA (Principal Component Analysis) . Also printing the score (Contribution of original dimension to the new dimension) pertaining to every new dimension.
* Given a dimension d, the goal is to reduce the input database in\_file.chst, in\_file.sift, in\_file.mvect to the reduced dimensional representation in new vector space using K-means. Also printing the score (Contribution of original dimension to the new dimension) pertaining to every new dimension.
* Given a video file vi and frame range [a,b] , an integer k and with the choice of distance/similarity methods, the goal is to find k - similar videos with the given frame range in the reduced dimension space (K-means reduced/ PCA - reduced).

Assumptions:

Assuming the input file database is given for histogram,sift and motion vector

Task 1 a-b histogram similarity:

* Assuming the files have same no of n-bins and cells with n-bin value = 10.
* Assuming all videos are in same directory.

Task 1 c-d SIFT similarity:

* Suppose you have a point P in Frame 1(F1) and you want to find the "best" match in Frame 2(F2). We compare the descriptor of P in F1 to all the descriptors in F2. For comparison we used Euclidean distance as suggested by Lowe in his paper “Distinctive Image Features from Scale-Invariant Key points” [1]. Then, we find two points in F2, say U & V which have the least and second-least distance (say, Du and Dv) from P respectively. Here's what Lowe recommended: if Dv/Du >= threshold (We used values between 1 - 1.5), then this match is acceptable; otherwise, it's ambiguously matched and is rejected as a correspondence and we don't match any point in F2 to P. Essentially, if there's a big difference between the best and second-best matches, you can expect this to be a quality match.

Task 1 e-f Motion vectors similarity:

* The Motion vector Similarity function requires that the motion vector (in\_file.mvect) file to be generated and the absolute file path for the same is known beforehand.
* The video files that need to be compared have to be provided.
* The type of distance similarity function that needs to be applied has to be provided.
* The mean slope (Four quadrant inverse tangent in radians) of every cell with respect to motion vector is calculated, ignoring all the motion vector with zero displacement.
* The slope values are taken from four quadrant inverse tangent (ranging from -pi to +pi)
* The slope is then transformed into new space by adding 2\*pi in order to avoid the slopes having 0 radian. So the new space for slope will range from pi to 3\*pi instead of -pi to +pi.

Task 2 a-b histogram subsequence:

* Assuming the files have same no of n-bins and cells with n-bin value = 10.
* Assuming all videos are present in same directory.
* Assuming we are not comparing same video.
* Assuming Query and sequence should be from database.

Task 2 c-d SIFT subsequence:

* Assuming that SIFT files for original dimensions are already given.
* All videos must be in same directory.
* Assuming all videos are present in same directory.
* Assuming Query and sequence should be from database.

Task 2 e-f Motion vectors subsequence:

* Assuming that motion vector files for original dimensions are already given.
* The Motion vector Similarity function requires that the motion vector (in\_file.mvect) file to be generated and the absolute file path for the same is known beforehand.
* The directory path of the videos is given.

Task 3:

Histogram:

* Assuming that in\_file.chst, in\_file.sift and in\_file.mvect files are already created for a given set of video files with R values as follows
  + - Histogram - 2
    - Sift - 2
    - Motion - 4
* The cells are taken as object and nBins as taken as feature/dimension in object-feature matrix. The PCA reduced vector space will be concentrated on the Bins with high covariance and the redundant bins will be truncated.
* Since subsequence search is to be made on the entire database, we are assuming the videos would be similar at some level, so the entire database containing the samples is transformed into new vector space. (Whole input database is passed as matrix)
* The original dimension is assumed to be 10 (Histogram nBins)
* The new\_dimension is numbered from 1 to d and original\_dimension(original\_index) is numbered from 1 to 10. The selected d dimensions are reported in the form of <new\_dimension,original\_index,score>

SIFT:

* The entire videos containing keypoint samples is taken as object and 128 descriptor values as feature in object-feature matrix.
* Since SIFT descriptors are invariant to uniform scaling, orientation and partially invariant to affine distortion, we are considering the 128 descriptors as the features in object-feature matrix which will be reduced to given dimension.
* The new\_dimension is numbered from 1 to d and original\_dimension (original\_index) is numbered from 1 to 128. The selected d dimensions are reported in the form of <new\_dimension,original\_index,score>

Motion vector:

* The frames are taken as objects and cells as feature in the object-feature matrix.
* Since the motion might be different for different videos, we are transforming object-feature matrix for individual videos. So each video will have different transformed space according to covariance of their respective cell across frames.
* D value should be less than the R\*R value (R\*R is the number of cells - original dimension, Here it is taken as 16 with R value being 4)
* The mean slope (Four quadrant inverse tangent in radians) of every cell with respect to motion vector is calculated, ignoring all the motion vector with zero displacement.
* The slope values are taken from four quadrant inverse tangent (ranging from -pi to +pi)
* The slope is then transformed into new space by adding 2\*pi in order to avoid the slopes having 0 radian. So the new space for slope will range from pi to 3\*pi instead of -pi to +pi.
* The cells with no motion displacement is taken as zero.
* The slope value alone is considered for motion vectors having displacement. Including distance measure or dot product would bring conflicts to the resultant values. Ex: If two motion vector with displacement 2(North direction) and 4(North West direction). Their slope values would be 1.57 and 0.78 respectively, so the arc length would becomes 3.14 and 3.14 for both case.
* The new\_dimension is numbered from 1 to d and original\_dimension(original\_index) is numbered from 1 to R\*R. The selected d dimensions are reported in the form of <video,new\_dimension,original\_index,score>

Description of proposed Solution:

Task 1(a,b):

Prerequisite

1. The Color Histogram algorithm requires that the Color Histogram data has to be generated and the absolute file path for the same is known beforehand.
2. The type of similarity function that needs to be applied has to be provided.
3. The directory where the video files are present needs to be provided to find the Videos in the input file.

Algorithm

Get the inputs for video name for video 1, video name for video 2 and similarity function from the user.

**Function Task\_1\_CHST()**

Read all the histogram data for video 1 and video 2

Copy the histogram for video 1 into one matrix and histogram data for video 2 into another matrix

Get the minimum frame count between the number of frames in the 2 videos by min(video1framecount,video2framecount)

Compute the Intersection Similarity between the 2 videos by

For each frame in videos video1 and video2

For each cell in frame

For each bin in the cell of each frame

x= min(video1bin1,video2bin2)/max(video1bin1,video2bin2).

Sum=Sum+x

SimilarityPercentage=Sum/(number of frames\*number of cells)

Display the similarity percentage

Compute the Tanimoto Similarity between the 2 videos by

For each frame in videos video1 and video2

For each cell in frame

For each bin in the cell of each frame

Max=max(video1bin-i,video2bin-i)

x=(max(vid1bin1,vid2bin1)-min(vid1bin1,vid2bin2))/max(vid1bin1,vid2bin2).

Sum=Sum+x

SimilarityPercentage= Sum/(number frames\*number of cells);

Display the similarity value as a percentage.

End For

End For

End For

Implementation:

The Task\_1\_CHST function compares the given two video files, by calculating the similarity function for each cell of one frame of video1 with the corresponding cell of the frame in the other video. The functions Intersection and Tanimoto[10] similarity are used here.

Intersection Similarity; initially we form a vector using the magnitude of bins of the color histogram which is already generated. Using these vectors we get the average of min/max of the bins of color histogram.The individual cell values are summed up and the mean is taken.From this we get the percentage similarity between 2 frames. The Intersection similarity of a perfectly matched frame is 1 and that of 2 frames with no match is 0.

Tanimoto Similarity; initially we form a vector using the magnitude of bins of the color histogram which is already generated. Using these vectors we get the average of (max-min)/max of the bins of color histogram.The individual cell values are summed up and the mean is taken.From this we get the percentage similarity between 2 frames. The Tanimoto similarity of a perfectly matched frame is 0 and that of 2 frames with no match is 1.

Correctness:

Given a pair of videos, it will output similarity percentage and it has to be verified manually by verifying the given video files.

Output:

The percentage of similarity between the 2 videos.

Sample Output: 94.32%

Task 1 (c,d)

Prerequisite:

1.The SIFT algorithm requires that the SIFT file has to be generated and the absolute file path for the same is known beforehand.

2.The video files that needs to be compared has to be provided.

3.The type of distance similarity function that needs to be applied has to be provided.

4.The threshold for the descriptor similarity has to be hardcoded before executing the task.

Algorithm:

SIFTSimilarity( ) function

Get the inputs for video1, video2, type (type of distance function)

Read the video data from the given SIFT file into matrix V1, V2

for each frame in min ( frames in V1 , frames in V2 )

generate matrixVideo1Framei and matrixVideo2Framej

distanceMatrix= EucledianDistance( matrixVideo1Framei, matrixVideo2Framej);

distanceMatrix= ManhattanDistance( matrixVideo1Framei, matrixVideo2Framej);

for each descrisptor ( dk ) in distanceMatrix

if( secondmin(dk)/ firstmin(dk) >= 1.5)

// its a match

DescriptorMatchedCount++;

end if;

end for;

end for;

totalPercentageMatch = descriptorMatchedCount/totalDescriptorsofTheMinimumFrameMatched

EuclideanDistance( ) function

Given two frame Matrix matrixVideo1Framei,matrixVideo2Framej

For each row in framei

For each row in framej

// Calculate the Euclidean distance and

// store the distance value in Matrix[framei,framej]

End for;

End For;

ManhattanDistance( ) function

Given two frame Matrix matrixVideo1Framei,matrixVideo2Framej

For each row in framei

For each row in framej

// Calculate the Manhattan distance and

// store the distance value in Matrix[framei,framej]

End for;

End For;

Implementation:

The SIFTSimilarity function compares the given two video files, by comparing the descriptor of one frame of video1 with all the descriptors of corresponding frame in the other video. Based on Lowes assumption, if the secondmin/firstmin ≥ 1.5 of a descriptor, the descriptor is considered as a good match otherwise neglected. The functions Euclidean distance and Manhattan distance basically finds the distance between the descriptors in the corresponding frames. The total number of descriptors matched across frames divided by the total descriptors results in the match percentage.

Correctness:

Given a pair of video files, it will output similarity percentage and it has be verified manually by verifying the given video files.

Output sample:

Similarity Percentage is : 66.70

Task 1(e,f):

Prerequisite:

1. For Intersection similarity we create a vector whose features are the magnitude and slope angle of the motion vector blocks.

2. When calculating the average slope angle; we represent the angle in between 2 Pi and 4 Pi. Otherwise using the range 0 to 2 Pi could lead to ambiguity if there is no motion vector present, resulting in zero same as if there is no difference in the motion vectors between 2 frames of different videos.

Pseudo-code:

% Get the inputs for video1, video2, type( type of similarity function)

% Read the video data from the given .mvect file into matrix V1, V2

MotionVectorSimilarity( ) function

for each frame in min ( frames in V1 , frames in V2 )

Generate matrixVideo1Framei and matrixVideo2Framej

averageDistanceMatrix = averageDistanceMatrix + ManhattanDistance(matrixVideo1Framei,matrixVideo2Framej);

endfor;

totalPercentageMatch = averageDistanceMatrix / min ( frames in V1 , frames in V2 )

% Given two frame Matrix matrixVideo1Framei, matrixVideo2Framej

ManhattanDistance ( ) function

For each cell in frame\_i, frame\_j

% Get the manhattan distance for cell\_i\_a, cell\_j\_b

End For;

IntersectionSimilarity ( ) function

For each cell in frame\_i, frame\_j

% Get the intersection similarity for cell\_i\_a, cell\_j\_b

endfor;

Implementation:

The MotionVectorSimilarity function compares the given two video files, by calculating the similarity function for each cell of one frame of video1 with the corresponding cell of the frame in the other video. The functions Manhattan distance and Intersection similarity are used here.

Manhattan distance; we get the average slope of the motion directions for the cells in a frame. Then take the absolute of the difference between the slopes for the corresponding cells of each frame of the two videos. The individual cell values are then summed up and the mean is taken using the number of cells in a frame. From this we get the percentage similarity between 2 frames.

Intersection similarity; initially we form a vector using the magnitude of the motion vector, this is calculated using Euclidean distance, and we take the slope angle as direction. Using these vectors of each cell we get the average of the ratio of these vectors for the cells in a frame. The individual cell values are then summed up and the mean is taken using the number of cells in a frame. From this we get the percentage similarity between 2 frames.

Correctness:

Given a pair of video files, it will output similarity percentage and it has to be verified manually by verifying the given video files.

Output sample:

Similarity Percentage using Manhattan distance is : 66.70

Similarity Percentage using Intersection similarity is : 73.69

Task 1(g,h):

Prerequisite:

1.The algorithm requires that the SIFT,CHST,MVECT file has to be generated and the absolute file path for the same is known beforehand.

2.The video files that needs to be compared has to be provided.

3.The type of distance similarity function that needs to be applied has to be provided.

Algorithm:

1.Calculate the Color Histogram Similarity , SIFT Similarity and Motion Vector Similarity from the functions

2.Geometric mean of percentages obtained from the above similarity or Harmonic Mean of percentages obtained from the above similarity.

Implementation:

We calculate the similarity percentage for Color Histogram,SIFT, Motion Vector Similarity. Based on the user input , we calculate either geometric mean or the harmonic mean.

Geometric mean is the nth ( i.e 3 here ) root of the product of similarity percentages obtained for each vector space ( Color Hisotgram, SIFT, Motion Vector ).

Harmonic mean is obtained using harmmean inbuilt function of MATLAB. The function takes in the three percentage similarity ( Color Hisotgram, SIFT, Motion Vector )

as the vector and gives the harmonic mean of them.

Correctness:

Given a pair of video files, it will output the overall similarity percentage using any of the methods mentioned above and it has be verified manually by verifying the given video files.

Output sample:

Overall Similarity Percentage is : 88.70

Task 2(a,b)/Task 4(a,b):

Prerequisite

1. The Color Histogram algorithm requires that the Color Histogram data has to be generated and the absolute file path for the same is known beforehand
2. The query video file that needs to be compared to generate the k-most similar subsequence has to be provided.
3. The type of similarity function that needs to be applied has to be provided
4. The value for k (no. of similar subsequences) has to be provided as input.
5. The directory where the video files are present needs to be provided to generate the k most similar subsequence videos.
6. The sequence range for which subsequences needs to be generated has to be provided as the input. The frame sequence assumed to be within the given files max frame count.
7. The number of dimensions of the histogram data in the reduced space must be known.

Algorithm

Get the inputs for path of the directory where all the videos are found, the query video name, starting frame of the query sequence, final frame of the query sequence, similarity measure, number of most similar sequences to be generated (and number of dimensions of bins in reduced space in the case of task 4).

**Function Task\_2\_CHST()/ Task\_4\_CHST()**

Read histogram data that is already generated for all the videos in the directory to a VideoData matrix (and get the number of bins in the case of task 4 based on the number of dimensions parameter input).

From the VideoData matrix copy the frames pertaining to query data

For video=queryVideoName

For i=starting frame of query sequence to ending frame of query sequence

Copy the histogram data to a queryVideoData Matrix

End For

End For

For each video in the directory except the query video

For currentframe=1 to max frame in the video

For i=0 to number of frames in query sequence-1

Copy the histogram data to of (currentframe +i) to MatrixVideoData matrix

Compute Intersection similarity or Tanimoto similarity between current sequence and query as similarityPercentMatrix(currentframe,1)=Similarity(MatrixVideo,queryVideo)

SimilarityPercentMatrix(currentframe,2)=current video name

SimilarityPercentMatrix(currentframe,3)=currentframe

End For

End For

End For

If similarity method = Intersection similarity

Sort SimilarityPercentMatrix descending based on Similarity percent

Else

Sort SimilarityPercentMatrix ascending based on Similarity percent

End If

For I=1 to k *//number of most similar sequences*

For each file, filename in directory

If SimilarityPercentMatrix(I,2)==filename

Read Video from dirPath

SequenceStart = SimilarityPercentMatrix(I,1)

SequenceEnd = SequenceStart + number of frames in querySequence -1

For Seq=SequenceStart to SequenceEng

Copy the Frame to a sequence video.

End For

End If

End For

End For

Display the k-most subsequences as video.

Implementation:

The Task\_2\_CHST and Task\_4\_CHST function compares the given two video files, by calculating the similarity function for each cell of one frame of video1 with the corresponding cell of the frame in the other video. The functions Intersection and Tanimoto similarity are used here.

Intersection Similarity; initially we form a vector using the magnitude of bins of the color histogram which is already generated. Using these vectors we get the average of min/max of the bins of color histogram.The individual cell values are summed up and the mean is taken.From this we get the percentage similarity between 2 frames. The Intersection similarity of a perfectly matched frame is 1 and that of 2 frames with no match is 0.

Tanimoto Similarity; initially we form a vector using the magnitude of bins of the color histogram which is already generated. Using these vectors we get the average of (max-min)/max of the bins of color histogram.The individual cell values are summed up and the mean is taken.From this we get the percentage similarity between 2 frames. The Tanimoto similarity of a perfectly matched frame is 0 and that of 2 frames with no match is 1.

Correctness:

Given a pair of videos, it will output similarity percentage and it has to be verified manually by verifying the given video files.

Output:

The k-most similar subsequences, each visualized by a video.

Sample Output: <Screenshot>

Task 2 (c,d)/ Task 4(c,d):

Prerequisite:

1.The SIFT algorithm requires that the SIFT file has to be generated and the absolute file path for the same is known beforehand.

2.The query video file that needs to be compared to generate the k most similar subsequence has to be provided.

3. The value for k (no. of similar subsequences ) has to be provided as input.

4.The type of distance similarity function that needs to be applied has to be provided.

5.The threshold for the similarity has to be hardcoded before executing the task.

6. The directory where the video files are present needs to be provided to generate the k most similar subsequence videos.

7. The sequence range for which subsequence needs to be generated has to be provided as the input. The frame sequence assumed to be within the given video files max frame count.

8. For Task - 4 the only change in the algorithm is we take in reduced dimension as an input.

Algorithm:

Task2\_SIFT( ) / Task4\_SIFT function

Get the inputs for directory Path for videos, query video file name, sequence [ a , b] ,type( type of distance function) , k ( no. of similar subsequence ) and reduced dimension d ( if Task – 4).

Read the video data from the given SIFT file into matrix VideoMatrix

For each video vi in the videoList

For each query frame qi of the given subsequence

for each frame fi in video vi

generate matrixQueryFrame qi and matrixVideoFrame fi

distanceMatrix= EucledianDistance(matrixQueryFrame, matrixVideoFrame)/ ManhattanDistance(matrixQueryFrame, matrixVideoFrame);

for each descriptor ( dk ) in distanceMatrix

if( secondmin(dk)/ firstmin(dk) >= 1.5)

// its a match

descriptorMatchedCount++;

end if;

end for;

percentageMatchPerFrame = descriptorMatchedCount/totalDescriptorsofTheMinimumFrameMatched;

//Calculate the query frame and the video frame similarity and store in the matrix

similarity\_percentage\_matrix ( qi,fi) = percentageMatchPerFrame;

end for;

end for;

// Calculate the subsequence percentage form the similarity\_percentage\_matrix and store it in k\_most\_similar matrix

For each frame fi in max(framecount of vi )

Offset = 0;

For each consecutive frames fj till ( total frame in vi - query\_frame\_range(b-a +1) )

SimialrityScore=SimilarityScore + similarity\_percentage\_matrix ( a + offset , fj )

offset + 1;

End for;

AverageSimilarityScore = SimialrityScore / query\_frame\_range( # of frames );

// store the result in k\_most\_similar matrix

k\_most\_similar matrix = ( vi , AverageSimilarityScore,fi to fj )

end for;

sorted\_k\_most\_similar = sort ( k\_most\_similar matrix ); // based on the percentage score

for first k most similar

// read the video and corresponding frames from the directory and create a new video subsequence

writeVideo(writerObj, currentFrameGray);

end for;

Implementation:

The Task2\_SIFT/Task4\_SIFT function compares the given query video with all the other videos in database except the given query video. It starts by comparing the descriptor of one frame of query video with all the descriptors of corresponding frame in the other videos in database. Based on Lowe’s assumption, if the secondmin/firstmin ≥ 1.5 of a descriptor, the descriptor is considered as a good match otherwise neglected. The functions Euclidean distance and Manhattan distance basically finds the distance between the descriptors in the corresponding frames. The total number of descriptors matched across frames divided by the total descriptors in that frame results in the match percentage which is stored in matrix A. The generated matrix A across all frames is checked for subsequences and an average percentage is calculated for each subsequence and stored in matrix B which contains the video number, average percentage and the frame sequence. The matrix B is sorted according to average percentage and a video sequence is generated for the k most sequences provided by the user.

Correctness:

The subsequence videos generated for query video has to be manually verified to find the correctness. The subsequence videos generated have so far been the best matches.

Output sample:

Given a query video vq, the program generates k most similar video sequences ( mp4 files ) in the same directory where the all the database videos exist.

Task 2 (e,f)/ Task 4 (e,f):

Prerequisite:

1. For Intersection similarity we create a vector whose features are the magnitude and slope angle of the motion vector blocks.

2. When calculating the average slope angle; we represent the angle in between 2 Pi and 4 Pi. Otherwise using the range 0 to 2 Pi could lead to ambiguity if there is no motion vector present, resulting in zero same as if there is no difference in the motion vectors between 2 frames of different videos.

Pseudo-code:

% Get the inputs for video index, frame range, (k) number of most similar sequences, type( type of similarity function)

% Read the video data from the given .mvect file into matrix V1, V2

MotionVectorSubSequence( ) function

% Using queue to store k most similar sub sequence videos. The queue matrix has 3 columns – video % index, video frame index, average similarity percentages for consecutive frames.

for each frame in min ( frames in V1 , frames in V2 )

Generate matrixVideo1Framei and matrixVideo2Framej

similarityPercentage = similarityPercentage + ManhattanDistance(matrixVideo1Framei,matrixVideo2Framej);

if (similarityPercentage > K\_sim\_matrix(1st\_row, avg\_similarity\_score))

% Store the video index, frame index and average score. Then cascade the current (k - 1)

% scores down.

endif

endfor;

% Visualize the query and k- subsequent videos.

% Given two frame Matrix matrixVideo1Framei, matrixVideo2Framej

ManhattanDistance ( ) function

For each cell in frame\_i, frame\_j

% Get the manhattan distance for cell\_i\_a, cell\_j\_b

End For;

IntersectionSimilarity ( ) function

For each cell in frame\_i, frame\_j

% Get the intersection similarity for cell\_i\_a, cell\_j\_b

endfor;

Implementation:

The MotionVectorSubSequence function compares the given video of certain sub-sequence with the remaining videos files. It compares the motion vector data for all the subsequences of all the videos with the data from the query video. For each sub-sequence we get an average similarity score which is stored in a queue if the value is large enough. The functions used to get the similarity scores are Manhattan distance and Intersection similarity are used here.

Manhattan distance; we get the average slope of the motion directions for the cells in a frame. Then take the absolute of the difference between the slopes for the corresponding cells of each frame of the two videos. The individual cell values are then summed up and the mean is taken using the number of cells in a frame. From this we get the percentage similarity between 2 frames.

Intersection similarity; initially we form a vector using the magnitude of the motion vector, this is calculated using Euclidean distance, and we take the slope angle as direction. Using these vectors of each cell we get the average of the ratio of the minimum vector to the maximum vector for the cells in a frame. The individual cell values are then summed up and the mean is taken using the number of cells in a frame. From this we get the percentage similarity between 2 frames.

Correctness:

Given a pair of video files, it will output subsequent videos which need to be compared manually with the query video.

Output sample:

<Screenshot at the end>

Task 3\_1:

Algorithm:

PCA - Histogram:

Get the input database in\_file.chst,Target dimensionality-D value,

Read the data from the file in\_file.chst and form a matrix X

Matrix X holds nBin values for all cells across videos

Perform PCA on Matrix X - returns M (Eigen vector) and Projection (New vector space)

For each column in M

Sort the values and the Original\_index together

End

write the Projection matrix to a file out\_file\_d.cpca

write the Original\_index,score to a file out\_file\_d\_score.cpca

Implementation:

The algorithm takes the in\_file.chst which is the histogram values of 10 bins (default). Reads the target dimensionality value ‘D’ from user. A matrix ‘X’ is constructed considering only the nBin values for the entire database. The reason for taking the entire database is because subsequence search is to be made on the entire database, we are assuming the videos would be similar at some level, so the entire database containing the samples is transformed into new vector space. (Whole input database is passed as matrix). The matrix ‘X’ is passed to the pca library function to obtain the reduced vectors and the matrix which contains the contribution of original dimension to the new dimension

The algorithm is implemented in Matlab IDE. The output is given of the format <VideoFile; Frame; Cell; Reduced Histogram values in new space>. Also the ‘D’ dimensions are reported in the form of <Di,Original\_index,score>. The following libraries are used for the required purpose.

[coeff,projection] = pca(X) - principal component ‘projection’ are the representations of X in the principal component space .

Output Sample:

<1;1;1;-29.375671,3596.065570,-152.999130,2.450422,0.373374,1.997806,-0.700568,> - Reduced to 7 dimension

Sample Scores for 1st new dimension(D1) sample : 10 Old dimension

<1;10;9.464247e-01><1;7;1.167290e-01><1;4;1.160344e-01><1;6;1.143870e-01><1;5;1.126562e-01><1;9;1.120053e-01><1;2;1.117027e-01><1;8;1.114164e-01><1;3;1.111263e-01><1;1;4.036745e-02>

PCA - SIFT:

Get the input database in\_file.sift,Target dimensionality-D value,

Read the data from the file in\_file.sift and form a matrix X

Matrix X holds 128 descriptor values for all cells across videos

Perform PCA on Matrix X - returns M (Eigen vector) and Projection (New vector space)

For each column in M

Sort the values and the Original\_index together

End

write the Projection matrix to a file out\_file\_d.spca

write the Original\_index,score to a file out\_file\_d\_score.spca

Implementation:

The algorithm takes the in\_file.sift which is the descriptor values of Sift vector - 128 feature (default). Reads the target dimensionality value ‘D’ from user. A matrix ‘X’ is constructed considering only the 128 descriptor values for the entire database. The reason for taking the entire database is because subsequence search is to be made on the entire database, we are assuming the videos would be similar at some level, so the entire database containing the samples is transformed into new vector space. (Whole input database is passed as matrix). The matrix ‘X’ is passed to the pca library function to obtain the reduced vectors and the matrix which contains the contribution of original dimension to the new dimension

The algorithm is implemented in Matlab IDE. The output is given of the format <VideoFile; Frame; Cell;x,y,scale,orientation; Reduced Sift descriptor values in new space>. Also the ‘D’ dimensions are reported in the form of <Di,Original\_index,score>. The following libraries are used for the required purpose.

[coeff,projection] = pca(X) - principal component ‘projection’ are the representations of X in the principal component space .

Output Sample:

<1;1;3;22;156;2.120821e+00;3.084898e+00;[-0.392294,-0.131030,0.122500,-0.174974,0.209968,-0.170433,0.080625,0.264407,-0.140573,-0.129887,-0.126100,0.139050,0.024972,0.002240,-0.210213,-0.032967,-0.073610,0.105814,-0.024647,0.024013,-0.066572,0.016594,0.045444,-0.097997,0.052209,0.060759,-0.000752,0.068976,-0.030018,-0.086743,-0.000352,-0.000479,-0.032044,0.019590,-0.020042,0.037160,0.048412,-0.058694,0.075290,0.014663,0.092156,0.007227,-0.050956,-0.023680,-0.062974,0.095101,0.004575,-0.001882,-0.016230,-0.045072,0.015091,0.003922,-0.030167,-0.060441,-0.003753,0.031942,0.036876,0.032940,0.009641,0.005051,0.012345,-0.043190,-0.023395,-0.012383,-0.035303,0.013484,-0.017625,0.017243,0.017475,0.031418,0.035058,0.029788,-0.015941,0.056216,-0.010009,-0.012247,0.005201,-0.008439,-0.029380,0.004049,]>

Sample Scores for 1st new dimension(D1) sample : 128 Old dimension

<1;71;2.634026e-01><1;35;2.628337e-01><1;91;2.610740e-01><1;63;2.610270e-01><1;79;1.869441e-01><1;43;1.865883e-01><1;55;1.823637e-01><1;83;1.823169e-01><1;113;1.632935e-01><1;123;1.627340e-01><1;9;1.618926e-01><1;3;1.617860e-01><1;17;1.579999e-01><1;105;1.566342e-01><1;117;1.548314e-01><1;21;1.538186e-01><1;13;1.514543e-01><1;109;1.506418e-01><1;31;1.470640e-01><1;103;1.458081e-01><1;41;1.307806e-01><1;81;1.299848e-01><1;73;1.239880e-01><1;49;1.230268e-01><1;23;1.196641e-01><1;115;1.182614e-01><1;111;1.178588e-01><1;11;1.160138e-01><1;97;1.100213e-01><1;1;1.093105e-01><1;25;1.063094e-01><1;121;1.055620e-01><1;61;8.262646e-02><1;69;8.238756e-02><1;93;8.150097e-02><1;37;8.078507e-02><1;85;7.800069e-02><1;53;7.564369e-02><1;45;7.551727e-02><1;77;7.341693e-02><1;59;7.154982e-02><1;67;7.086153e-02><1;95;6.467999e-02><1;39;6.443302e-02><1;92;5.668755e-02><1;36;5.614966e-02><1;62;5.272074e-02><1;70;5.164507e-02><1;80;4.934669e-02><1;114;4.753474e-02><1;56;4.622775e-02><1;24;4.540151e-02><1;10;4.461250e-02><1;84;4.445716e-02><1;44;4.445012e-02><1;72;4.408691e-02><1;112;4.292948e-02><1;78;4.289344e-02><1;54;4.248981e-02><1;34;4.146541e-02><1;42;4.001920e-02><1;64;3.960953e-02><1;90;3.723102e-02><1;82;3.631303e-02><1;74;3.126294e-02><1;106;3.014603e-02><1;18;2.995791e-02><1;124;2.990750e-02><1;50;2.984325e-02><1;86;2.972534e-02><1;118;2.926688e-02><1;14;2.911710e-02><1;16;2.905104e-02><1;120;2.845867e-02><1;30;2.809672e-02><1;46;2.788637e-02><1;52;2.777397e-02><1;4;2.729835e-02><1;76;2.626908e-02><1;102;2.511464e-02><1;20;2.485688e-02><1;68;2.465957e-02><1;60;2.464620e-02><1;38;2.457760e-02><1;48;2.448132e-02><1;94;2.424596e-02><1;108;2.407223e-02><1;88;2.338902e-02><1;101;2.222223e-02><1;29;1.995797e-02><1;87;1.645438e-02><1;98;1.490299e-02><1;26;1.370727e-02><1;8;1.225029e-02><1;47;1.186217e-02><1;5;1.181322e-02><1;15;1.169231e-02><1;128;1.140599e-02><1;119;1.139820e-02><1;51;1.106600e-02><1;125;1.015603e-02><1;107;9.859817e-03><1;19;9.832554e-03><1;122;7.338886e-03><1;32;6.846778e-03><1;75;6.668023e-03><1;66;6.468313e-03><1;58;6.395993e-03><1;40;6.146772e-03><1;7;5.760384e-03><1;127;5.740868e-03><1;96;5.729989e-03><1;99;5.099909e-03><1;2;5.094674e-03><1;27;4.926871e-03><1;104;4.734987e-03><1;65;3.391075e-03><1;12;3.379916e-03><1;116;3.069644e-03><1;100;2.754289e-03><1;57;2.348192e-03><1;33;2.315274e-03><1;89;1.595246e-03><1;28;1.292275e-03><1;22;1.122129e-03><1;6;1.026337e-03><1;110;8.719263e-04><1;126;2.100668e-04>

PCA -Motion Vector:

Get the input database in\_file.mvect,Target dimensionality-D value,

Read the data from the file in\_file.mvect and form a matrix X row - frames, column - cell slope

Matrix X holds m frames and n cell slopes(motion) values for all frames in single video

Perform PCA on Matrix X - returns M (Eigen vector) and Projection (New vector space)

For each column in M

Sort the values and the Original\_index together

End

write the Projection matrix to a file out\_file\_d.mpca

write the Original\_index,score to a file out\_file\_d\_score.mpca

Implementation:

The algorithm takes the in\_file.mvect. Reads the target dimensionality value ‘D’ from user. A matrix ‘X’ is constructed considering average cell slope value as features and frames as object for every video. The slopes are calculated using four quadrant inverse tangent for every motion vectors which has displacement. The average slope is calculated. The reason for taking the individual video is because subsequence search is to be made on individual video’s newly transformed space based on the direction in which most of the vectors are moving. The matrix ‘X’ is passed to the pca library function to obtain the reduced vectors and the matrix which contains the contribution of original dimension to the new dimension

The algorithm is implemented in Matlab IDE. The output is given of the format <VideoFile; Frame;Reduced cells in new space>. Also the ‘D’ dimensions are reported in the form of <Vi,Di,Original\_index,score>. The following libraries are used for the required purpose.

[coeff,projection] = pca(X) - principal component ‘projection’ are the representations of X in the principal component space .

Output Sample:

Reduced from 16 Cells to 10 cells

<1;1;-1.816537e+01,-7.595860e-01,-3.736137e+00,-5.586637e-01,-4.255517e+00,-1.879213e+00,-9.260801e-02,9.746903e-02,2.579633e+00,-1.520940e-01,>

Sample Scores for 1st new dimension(D1) sample : 16 Old dimension

<1;1;15;2.840266e-01><1;1;13;2.814841e-01><1;1;14;2.814841e-01><1;1;2;2.814705e-01><1;1;1;2.814705e-01><1;1;9;2.733114e-01><1;1;10;2.726905e-01><1;1;12;2.678115e-01><1;1;16;2.657116e-01><1;1;11;2.647059e-01><1;1;6;2.444767e-01><1;1;5;2.441729e-01><1;1;3;2.255609e-01><1;1;8;1.965923e-01><1;1;4;1.472163e-01><1;1;7;1.018273e-01>

Kmeans - Histogram:

Get the input database in\_file.chst,Target dimensionality-D value,

Read the data from the file in\_file.chst and form a matrix X

Matrix X holds nBin values for all cells across videos

Perform Kmeans on Matrix X - returns M (Cluster centroid transpose) and Projection (Proximity to clusters)

For each column in M

Sort the values and the Original\_index together

End

write the Projection matrix to a file out\_file\_d.ckm

write the Original\_index,score to a file out\_file\_d\_score.ckm

Implementation:

The algorithm takes the in\_file.chst which is the histogram values of 10 bins (default). Reads the target dimensionality value ‘D’ from user. A matrix ‘X’ is constructed considering only the nBin values for the entire database. The reason for taking the entire database is because subsequence search is to be made on the entire database, we are assuming the videos would be similar at some level, so the entire database containing the samples is transformed into new vector space. (Whole input database is passed as matrix). The matrix ‘X’ is passed to the kmeans library function to obtain the reduced vectors and the matrix which contains the contribution of original dimension to the new dimension

The algorithm is implemented in Matlab IDE. The output is given of the format <VideoFile; Frame; Cell; Reduced Histogram values in new space>. Also the ‘D’ dimensions are reported in the form of <Di,Original\_index,score>. The following libraries are used for the required purpose.

[clustIDX, clusters, interClustSum, Dist] = kmeans(X, no\_dims);

‘clusters’ matrix gives the principal components

‘Dist’ matrix gives the representations of X in the principal component space .

Output Sample:

Reduced to 7 dimension

<1;1;1;52566599.871126,53840637.956509,733058.911533,51501051.717734,1156.846405,48057871.911373,317852.930409,>

Sample Scores for 1st new dimension(D1) sample : 10 Old dimension

<1;5;1.648652e+03><1;6;1.120570e+03><1;7;1.094142e+03><1;4;1.091346e+03><1;8;1.048804e+03><1;9;1.047607e+03><1;2;1.046411e+03><1;3;1.040511e+03><1;1;4.339320e+02><1;10;4.280243e+02>

Kmeans - SIFT:

Get the input database in\_file.sift,Target dimensionality-D value,

Read the data from the file in\_file.sift and form a matrix X

Matrix X holds 128descriptor values for all cells across videos

Perform Kmeans on Matrix X - returns M (transpose of clusters) and Projection (proximity to cluster centroid)

For each column in M

Sort the values and the Original\_index together

End

write the Projection matrix to a file out\_file\_d.ckm

write the Original\_index,score to a file out\_file\_d\_score.ckm

Implementation:

The algorithm takes the in\_file.sift which is the descriptor values of Sift vector - 128 feature (default). Reads the target dimensionality value ‘D’ from user. A matrix ‘X’ is constructed considering only the 128 descriptor values for the entire database. The reason for taking the entire database is because subsequence search is to be made on the entire database, we are assuming the videos would be similar at some level, so the entire database containing the samples is transformed into new vector space. (Whole input database is passed as matrix). The matrix ‘X’ is passed to the kmeans library function to obtain the reduced vectors and the matrix which contains the contribution of original dimension to the new dimension

The algorithm is implemented in Matlab IDE. The output is given of the format <VideoFile; Frame; Cell;x,y,scale,orientation; Reduced Sift descriptor values in new space>. Also the ‘D’ dimensions are reported in the form of <Di,Original\_index,score>. The following libraries are used for the required purpose.

[clustIDX, clusters, interClustSum, Dist] = kmeans(X, no\_dims);

‘clusters’ matrix gives the principal components

‘Dist’ matrix gives the representations of X in the principal component space .

Output Sample:

Reduced to 80 dimension

<1;1;3;22;156;2.120821e+00;3.084898e+00;[0.438436,0.885374,1.491348,1.049604,0.999014,0.568901,1.461566,0.822465,0.807132,0.942538,0.860057,0.827454,1.045912,1.355911,0.980755,1.041787,1.513573,1.197998,0.804355,0.981367,1.038420,0.959875,1.239819,1.567872,0.918681,0.833853,0.510960,1.291510,0.846551,0.409086,1.115636,0.998616,0.986145,1.532793,0.664158,0.832825,0.681136,1.145118,1.106107,0.813654,0.948828,0.990758,0.845227,0.842163,1.044515,0.792727,1.035068,0.972025,1.344629,1.045524,0.928873,1.068655,0.971655,1.298026,1.046046,0.812554,0.800487,1.531671,0.804390,0.831287,0.904278,1.487745,1.109893,0.833942,1.046684,1.069976,0.357646,1.195487,1.007046,1.566759,1.052063,1.002497,1.038230,1.017926,1.530788,1.051384,1.030164,0.989114,0.798876,0.640786,]>

Sample Scores for 1st new dimension(D1) sample : 128 Old dimension

<1;81;2.940522e-01><1;43;2.933339e-01><1;113;2.922940e-01><1;63;2.913023e-01><1;35;2.907796e-01><1;111;2.889333e-01><1;21;2.780706e-01><1;103;2.769908e-01><1;123;2.408419e-01><1;53;1.522075e-01><1;51;1.433592e-01><1;49;1.416095e-01><1;55;1.363077e-01><1;73;1.363006e-01><1;95;1.199625e-01><1;75;9.665548e-02><1;34;9.310236e-02><1;11;8.996988e-02><1;42;8.878679e-02><1;115;8.765920e-02><1;29;8.682960e-02><1;88;8.510773e-02><1;120;6.721524e-02><1;105;6.692703e-02><1;114;6.587630e-02><1;62;6.523225e-02><1;56;6.467970e-02><1;52;6.435925e-02><1;50;6.387750e-02><1;89;6.275373e-02><1;1;6.188542e-02><1;54;6.150438e-02><1;20;6.144360e-02><1;67;6.132104e-02><1;13;5.871265e-02><1;119;5.052120e-02><1;112;4.984045e-02><1;3;4.853485e-02><1;31;4.619436e-02><1;122;3.926162e-02><1;2;3.726840e-02><1;74;3.512685e-02><1;82;3.375855e-02><1;64;2.983811e-02><1;110;2.914230e-02><1;121;2.785569e-02><1;44;2.769197e-02><1;96;2.642545e-02><1;87;2.629008e-02><1;41;2.240153e-02><1;22;2.236279e-02><1;12;2.163896e-02><1;61;2.014598e-02><1;30;1.990389e-02><1;83;1.544856e-02><1;116;1.487592e-02><1;19;1.485708e-02><1;71;1.483018e-02><1;104;1.468980e-02><1;117;1.419295e-02><1;80;1.377990e-02><1;118;1.324571e-02><1;94;1.283863e-02><1;10;1.273546e-02><1;45;1.259425e-02><1;79;1.220256e-02><1;102;1.152636e-02><1;93;1.020614e-02><1;124;1.015984e-02><1;28;9.877296e-03><1;57;9.637377e-03><1;27;9.462743e-03><1;23;8.782557e-03><1;9;8.440569e-03><1;66;7.664151e-03><1;33;6.508871e-03><1;8;6.241889e-03><1;125;6.083506e-03><1;15;5.888129e-03><1;25;4.651434e-03><1;91;4.127273e-03><1;32;3.999889e-03><1;69;3.724521e-03><1;109;3.619762e-03><1;14;3.315862e-03><1;90;2.661719e-03><1;36;2.582678e-03><1;4;2.272278e-03><1;68;1.858381e-03><1;16;1.579641e-03><1;92;1.500978e-03><1;101;1.452842e-03><1;26;1.411146e-03><1;7;1.368478e-03><1;76;1.266193e-03><1;70;1.242324e-03><1;37;1.235813e-03><1;24;1.156697e-03><1;5;1.123622e-03><1;18;1.062531e-03><1;128;8.994988e-04><1;17;6.411582e-04><1;78;6.079236e-04><1;86;5.387913e-04><1;108;5.009352e-04><1;77;4.484243e-04><1;106;4.200973e-04><1;100;3.987994e-04><1;97;3.799820e-04><1;107;2.966837e-04><1;60;2.947817e-04><1;99;2.663925e-04><1;48;2.054174e-04><1;85;1.433154e-04><1;126;1.257917e-04><1;65;1.187640e-04><1;84;9.919100e-05><1;72;9.220689e-05><1;59;7.640378e-05><1;127;7.018975e-05><1;38;6.904016e-05><1;46;5.788950e-05><1;6;4.702517e-05><1;98;3.466351e-05><1;58;3.233274e-05><1;40;3.049607e-05><1;39;3.023152e-05><1;47;2.681703e-05>

Kmeans-Motion Vector:

Get the input database in\_file.mvect,Target dimensionality-D value,

Read the data from the file in\_file.mvect and form a matrix X row - frames, column - cell slope

Matrix X holds m frames and n cell slopes(motion) values for all frames in single video

Perform Kmeans on Matrix X - returns M (cluster centroid) and Projection (Proximity to the clusters)

For each column in M

Sort the values and the Original\_index together

End

write the Projection matrix to a file out\_file\_d.mpca

write the Original\_index,score to a file out\_file\_d\_score.mpca

Implementation:

The algorithm takes the in\_file.mvect. Reads the target dimensionality value ‘D’ from user. A matrix ‘X’ is constructed considering average cell slope value as features and frames as object for every video. The slopes are calculated using four quadrant inverse tangent for every motion vectors which has displacement. The average slope is calculated. The reason for taking the individual video is because subsequence search is to be made on individual video’s newly transformed space based on the direction in which most of the vectors are moving. The matrix ‘X’ is passed to the kmeans library function to obtain the reduced vectors and the matrix which contains the contribution of original dimension to the new dimension

The algorithm is implemented in Matlab IDE. The output is given of the format <VideoFile; Frame;Reduced cells in new space>. Also the ‘D’ dimensions are reported in the form of <Vi,Di,Original\_index,score>. The following libraries are used for the required purpose.

[clustIDX, clusters, interClustSum, Dist] = kmeans(X, no\_dims);

‘clusters’ matrix gives the principal components

‘Dist’ matrix gives the representations of X in the principal component space .

Output Sample:

Reduced to 10 dimensions

<1;1;8.796488e+02,1.707371e+02,3.597585e+01,3.237829e+02,6.894462e+01,6.548613e+02,6.321467e+02,9.730994e+02,1.108830e+02,4.907634e+02,>

Sample Scores for 1st new dimension(D1) sample : 16 Old dimension

<1;1;16;7.889434e+00><1;1;1;7.850796e+00><1;1;2;7.850796e+00><1;1;3;7.850796e+00><1;1;4;7.850796e+00><1;1;5;7.850796e+00><1;1;10;7.752622e+00><1;1;8;7.458097e+00><1;1;9;7.458097e+00><1;1;7;7.421347e+00><1;1;6;7.327198e+00><1;1;11;7.078607e+00><1;1;13;6.991898e+00><1;1;14;6.991898e+00><1;1;15;6.991898e+00><1;1;12;5.687731e+00>

User Interface specification:

Change History:

Open Issues from Phase 1 is resolved. Currently the project has no open issues.

Resolved Issue: For different videos having the motion vectors outside the frame's column is not resolved. Note: The motion vectors lying outside the frame's row has been resolved.

Display description:

The user can visualize the k most similar sequence of videos in the given Database.

System requirements/installation:

Matlab 2016

Visual Studio 2015 or any equivalent IDE

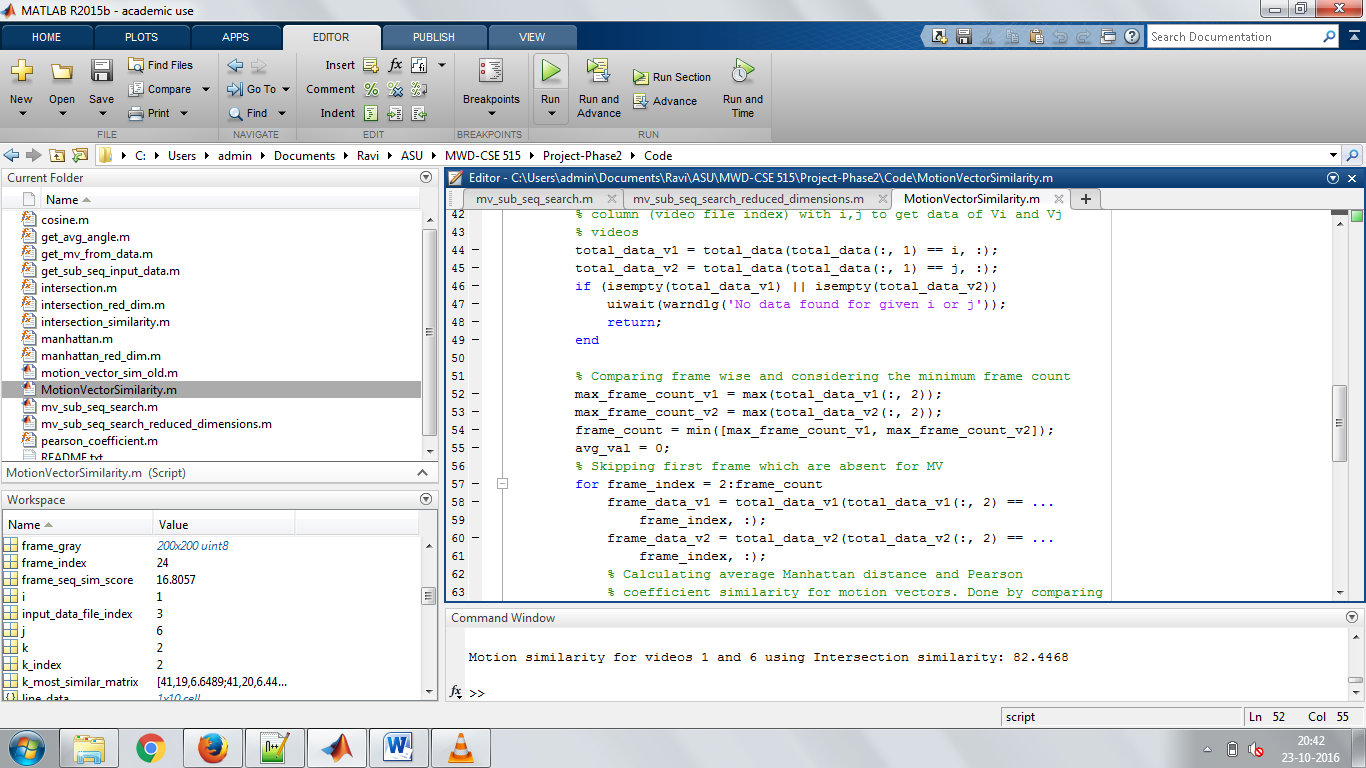
Related Works:

There are many approaches to measure the video similarity and video classification. One popular video representation technique is to represent each video sequence with frames [2]. Recently, the technique for measuring the video similarity based on the percentage of visually similar frames between the two sequences has been proposed in [1],[3],[4]. One commonly used technique for video similarity measurement is the Naıve Video Similarity (NVS) [1],[3]. This technique first finds the total number of frames from each video sequence which has at least one similar frame with the other sequence. Then, the ratio of these numbers will be computed to the total numbers of frames. After that, the threshold is used to compare the difference between the frames. Expectation-based measuring video similarity [5] can measure similarity of video efficiently by using expected value to average distance of video frames instead of the threshold. Each video sequence was represented with frame and each frame was represented with the color histogram to help enhance feature reduction. After that, categorization was performed using the nearest neighbor classifier with the 𝐿1 metric to measure distance by comparing each sampling frame of the training videos with all sampling frames of the test videos.

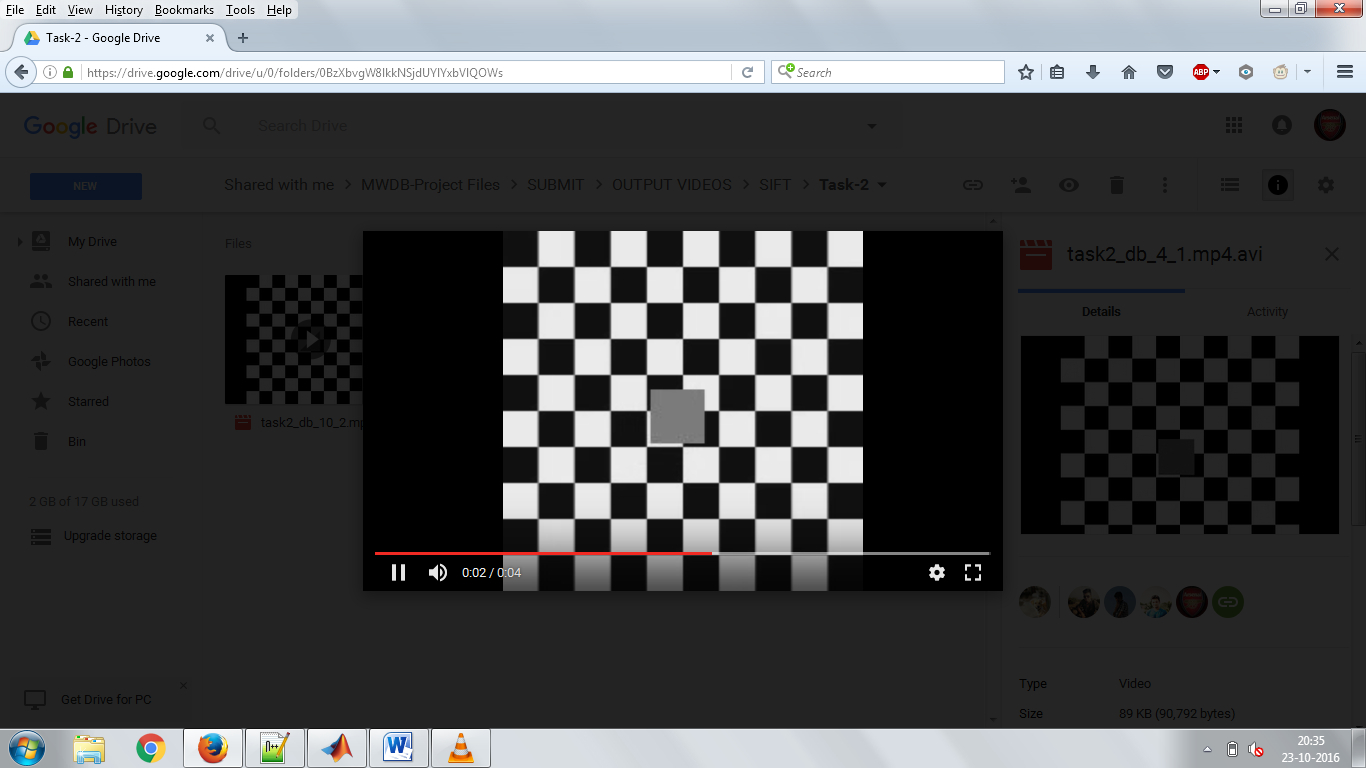
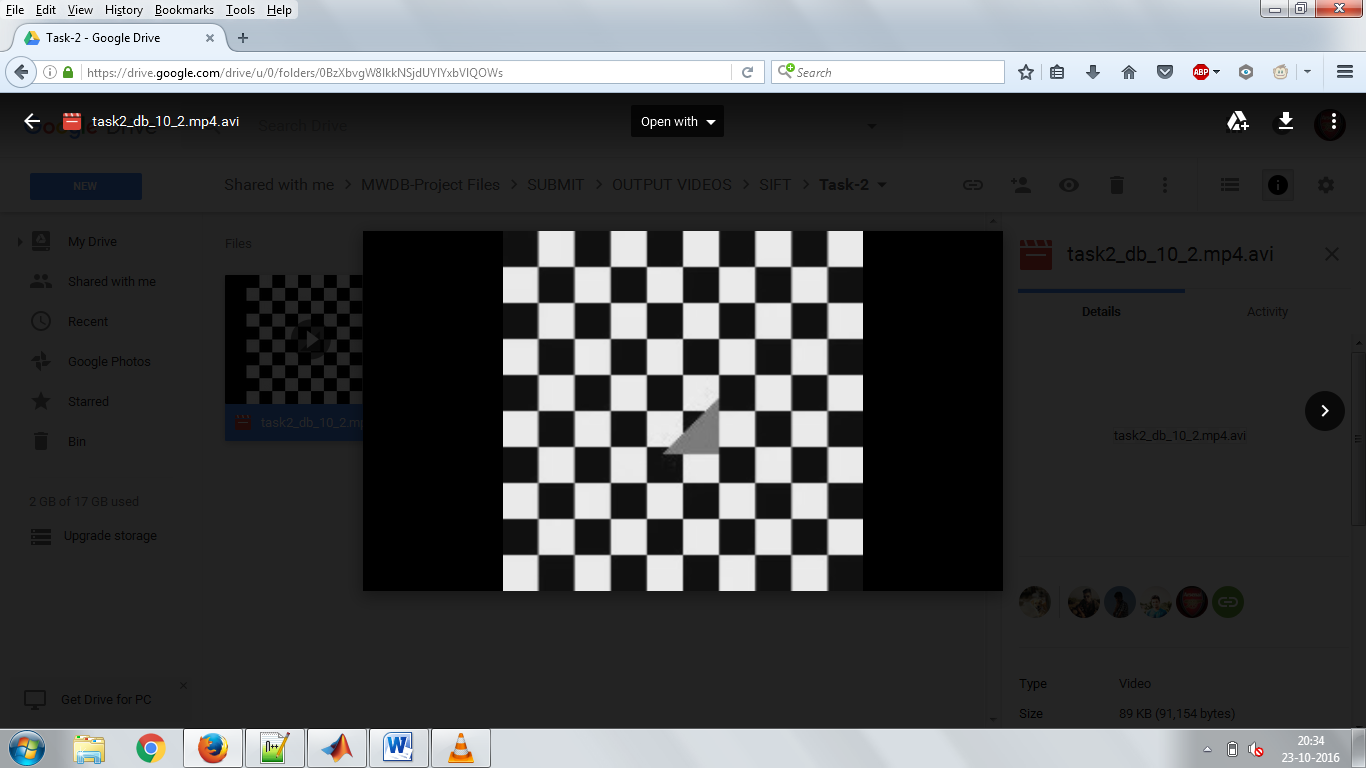
Principal component analysis (PCA) is a statistical procedure that transforms a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. The primary goal of PCA is to extract the dominant patterns exhibited in the data, making it an efficient technique for linear dimension reduction and a crucial preprocessing step in high dimensional statistical analyses and machine learning based classification or clustering. One of the major feature used to compare two images are SIFT key points. SIFT suffers from the so-called curse of dimensionality due to its high dimensional feature vectors. To overcome this weakness, we employ here the PCA-SIFT algorithm to project the high-dimensional samples onto a low-dimensional feature space using PCA. PCA-SIFT just takes the location, scale and dominant orientation of the key-point as its input feature vector. Eigen transformation required for the PCA algorithm is computed using the sample feature vectors, and the eigenvectors corresponding to the dominant eigenvalues constitute the basis vectors of the reduced feature space[6].

Clustering (a.k.a. unsupervised classification) is the task of categorizing data into groups without using any labels (training data), and it is omnipresent in various application domains such as data mining, pattern recognition, and signal processing [7]–[9]. Among numerous clustering algorithms, K-means is the most prominent one [7]–[9]. It thrives on “tight” groups of data points that can be linearly separated [via (hyper)planes], but it further has important extensions to kernel K-means, which can be rendered equivalent to spectral clustering – the popular graph-partitioning tool that can cope even with nonlinearly separable data points [10].

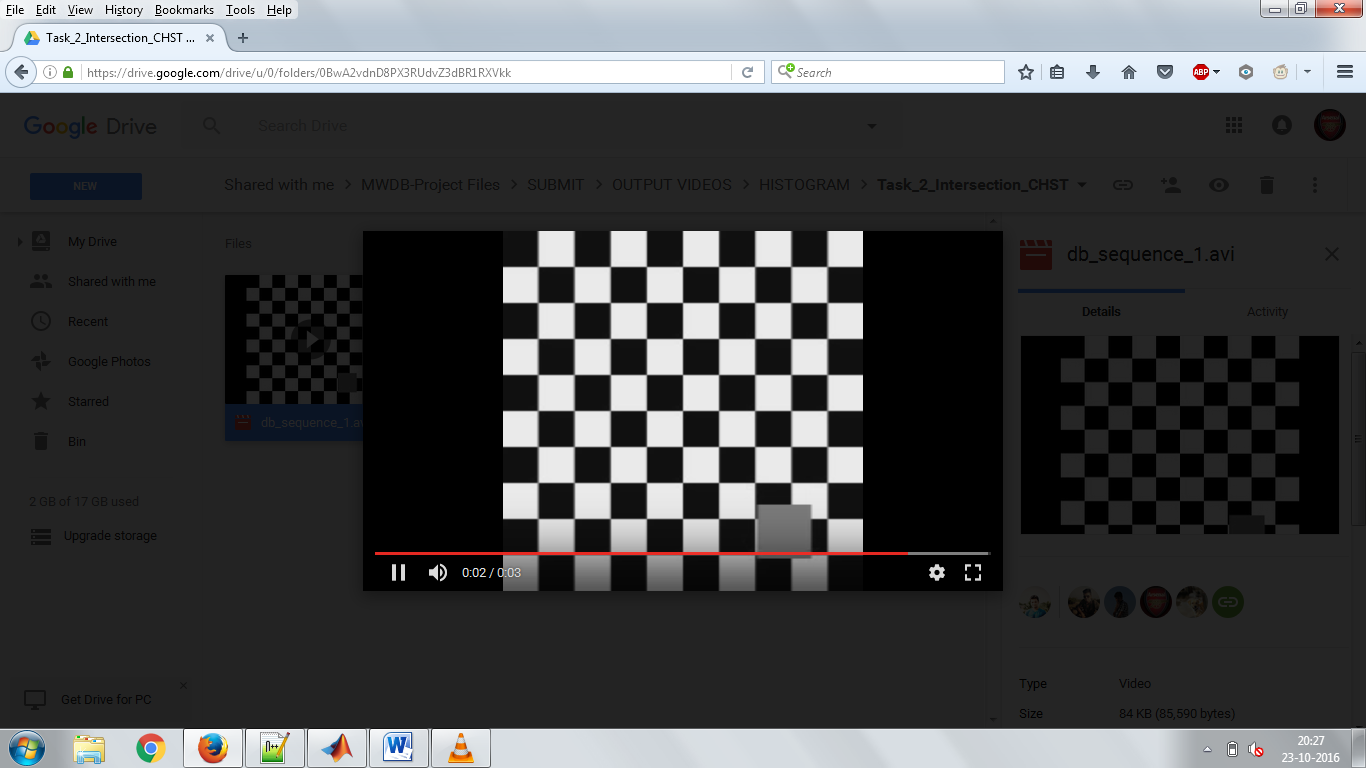
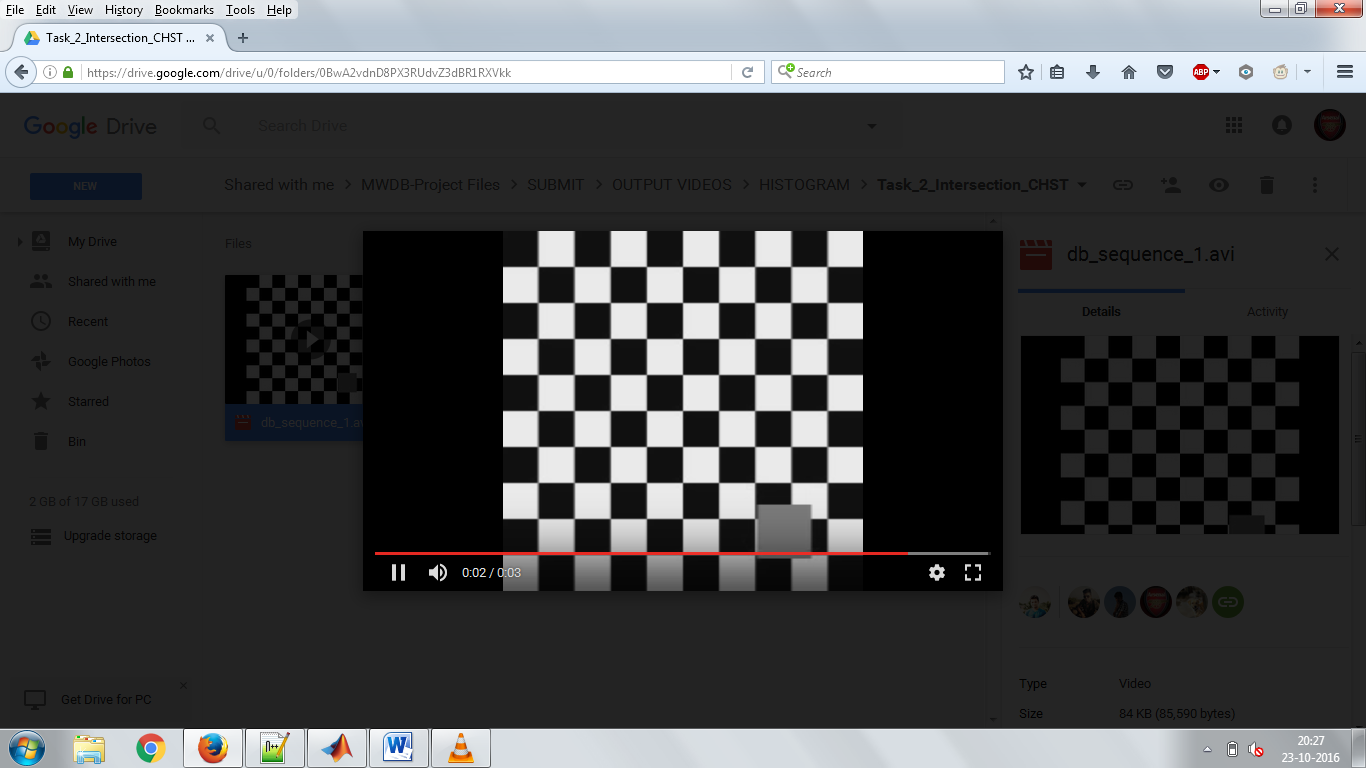
**Screenshots:**



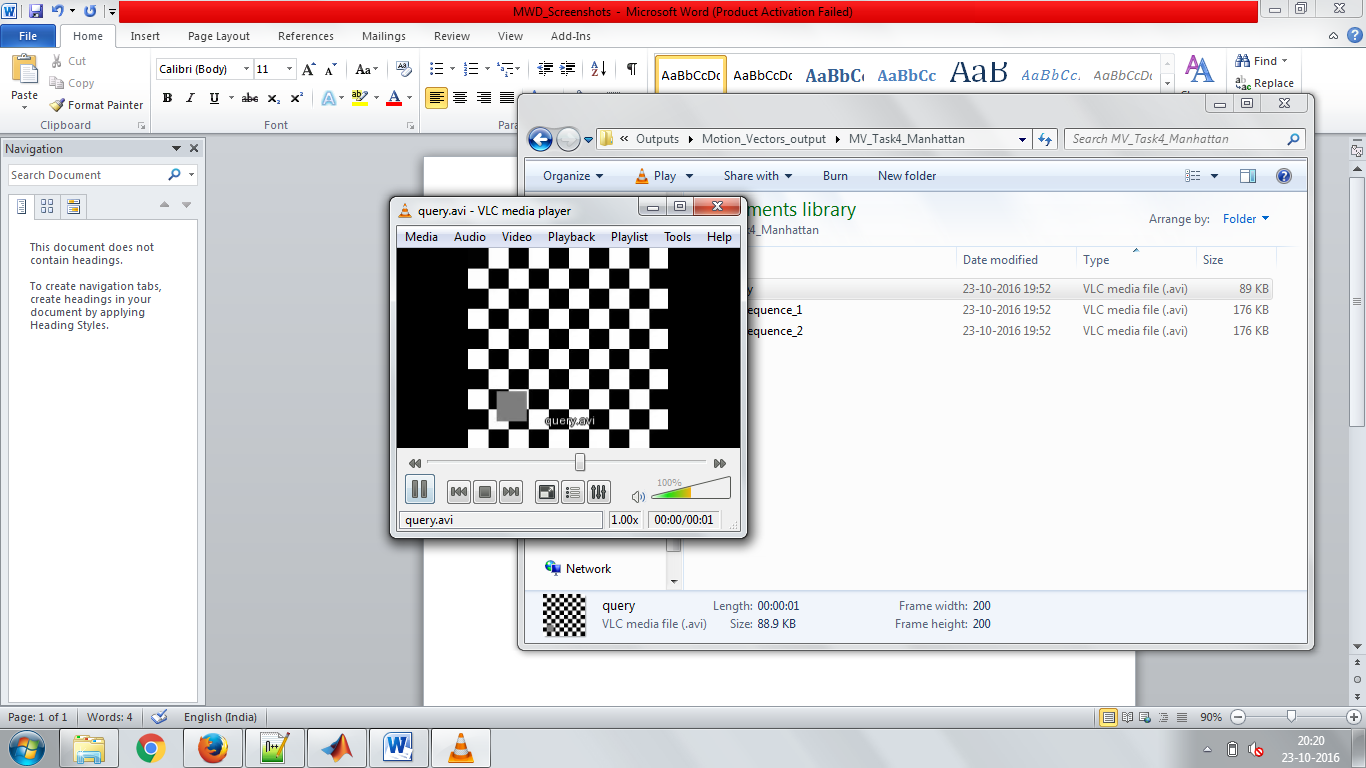
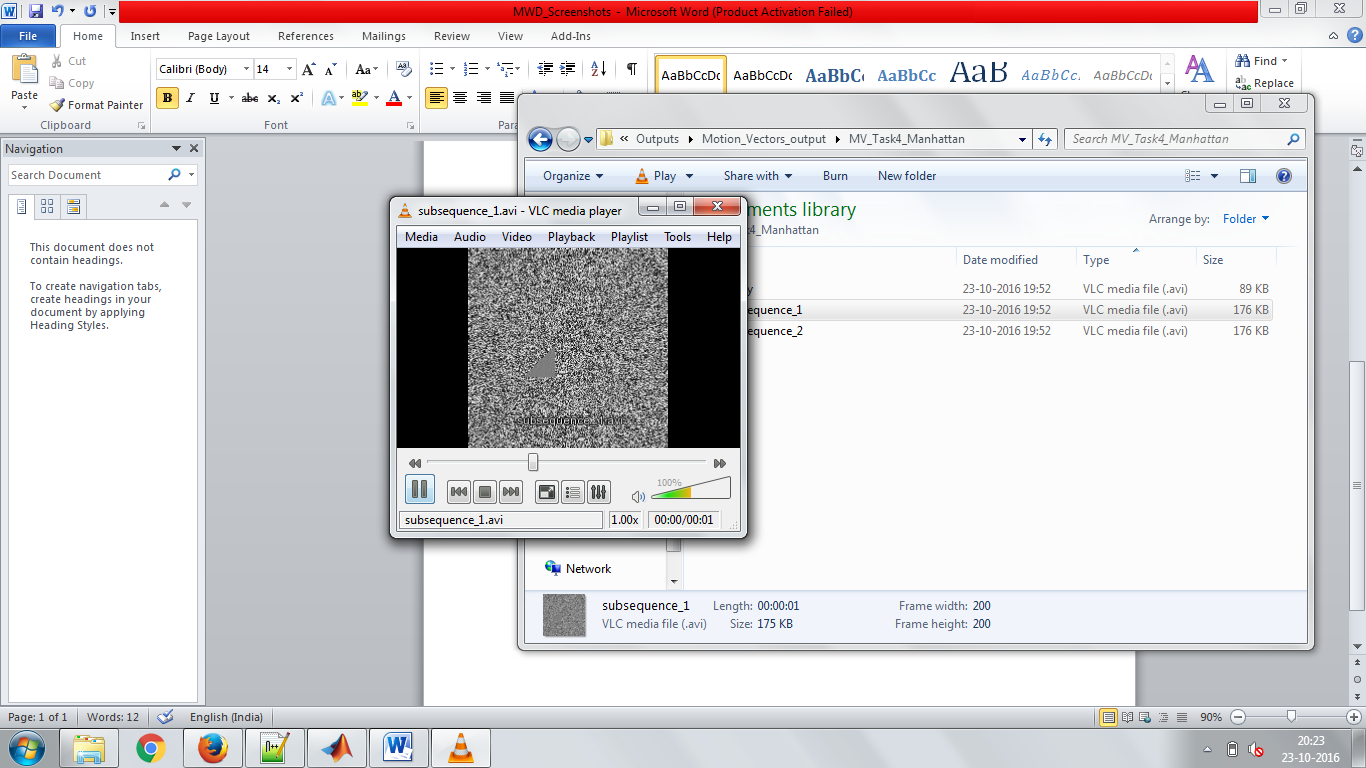
Motion similarity for videos 1 and 6 using Intersection similarity: 82.4468



Subsequence video on using Euclidean distance for SIFT descriptors. Left is the query video and right is the result subsequence.



Subsequence video on using Tanimoto similarity for Histogram. Left is the query video and right is the result subsequence.

Subsequence video on Reduced vectors using Manhattan distance for Motion Vectors. Left is the query video and right is the result subsequence.

|  |  |  |
| --- | --- | --- |
|  | Video\_1 and Video\_2 Similarity Score | Similarity Function |
| Histogram | 94.34 % | Tanimoto |
| SIFT | 89 % | Euclidean |
| Motion Vector | 82.45 % | Intersection |

Table showing results for Similarity scores of 2 videos.

Conclusion:

We have calculated similarity between the given two videos using similarity measures such as (Intersection similarity, Tanimoto similarity coefficient, Manhattan distance, Euclidean distance) and found the nearest k similar videos in the database in the nonreduced and reduced database obtained through PCA and kmeans. We also present results of the subsequence search in the database containing similar videos and visually represent the most similar videos based on the match percentage obtained from distance-similarity functions.

Bibliography:

[1] S.-S. Cheung and A. Zakhor, “Efficient video similarity measurement with video signature,” Circuits and Systems for Video Technology, IEEE Transactions on, vol. 13, no. 1, pp. 59 – 74, 2003.

[2] Xiangmin Zhou, Xiaofang Zhou, and Heng Tao Shen, “Efficient similarity search by summarization in large video database,” in ADC ’07: Proceedings of the eighteenth conference on Australasian database, 2007, pp. 161–167.

[3] S.-C.S. Cheung and A. Zakhor, “Fast similarity search on video signatures,” in Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on, 2003, vol. 2, pp. II – 1–4 vol.3.

[4] Heng Tao Shen, Beng Chin Ooi, and Xiaofang Zhou, “Towards effective indexing for very large video sequence database,” in SIGMOD ’05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, New York, NY, USA, 2005, pp. 730–741, ACM.

[5] P. Mutchima and P. Sanguansat, "Video similarity measurement approach via dimensionality reduction with distance space and random projection: Application with sports video classification," *Communications and Information Technologies (ISCIT), 2010 International Symposium on*, Tokyo, 2010, pp. 430-434. doi: 10.1109/ISCIT.2010.5664878

[6] Y. Shen, P. Guturu, T. Damarla, B. P. Buckles and K. R. Namuduri, "Video stabilization using principal component analysis and scale invariant feature transform in particle filter framework," in *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1714-1721, August 2009. doi: 10.1109/TCE.2009.5278047

[7] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. New York: Springer, 2001.

[8] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th ed. Academic Press, 2008.

[9] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification, 2nd ed. John Wiley & Sons, 2001.

[10] I. S. Dhillon, Y. Guan, and B. Kulis, “Kernel K-means: Spectral clustering and normalized cuts,” in Proc. SIGKDD. ACM, 2004, pp.551–556.

[11] <http://users.uom.gr/~kouiruki/sung.pdf>

[12] <https://www.quora.com/What-is-Tanimoto-Score-and-when-is-it-used>