Institute of Electrical Engineering

Signal Processing Laboratory LTS4

Analysis of Invariant Motion Components in Dynamic Video Content

Jose Molina

Assistant:
Ph.D. Jacob Chakareski

Supervisor:
Ph.D. Pascal Frossard

Index

I Introduction

I.1 Motivation

The fast expansion of Internet and DVB channels has brought a fast increase of video footage which needs to be indexed for efficient and easy retrieval. This task has been historically done by documentalists who tag manually each video with a few keywords, unfortunately such work is time consuming and hence very expensive. In the last decade much effort has been put into building processes which automatically assign content-based labels to video documents, a proof of this is the existence of the TRECVid Video Retrieval Evaluation [1] workshops since 2003.

I.2 Goals

Structural similarity metrics for still images has been largely studied lately, the goal of these is discovering underlying structure of an image that is impervious to rotations, translations, resizing and other transformation. This way images can be easily compared without being affected by their different scales and any kind of intermediate processing that they have experienced. The question tackled in this work is whether something analogous can be made for video, so similar videos can be detected independently of their size, frame rate and image content.

I.3 Related work

Video indexing for retrieval is an old concept, first approaches date from the first half of the nineties. Back in 1994, **Smoliar et al.** [2] already stated the necessity for video software to identify and represent video content for indexing and retrieval. In parallel to video indexing, video classification has also brought much attention, while retrieval focus on finding videos in a database that match a given query, classification puts all the input videos into predefined categories, which are labeled.

There are many ways to address these issues, mainly there have been three fields of research. One is text-based approach, which is based on identifying text objects and processing them with optical character recognition or extracting text from closed-captions, like did **Wei Qi et al.** [3] to automatically categorize news stories. Another approach is using audio features, processing the data can be made in time domain (energy, zero crossings): **E. Wold et al.** [4], **Z. Liu et al.** [5]; or in the frequency domain (bandwidth, frequency centroid, pitch): **U. Srinivasan et al.** [6]; a strong point in this case is the maturity of audio processing techniques. Finally, the third field of research is using visual information, which attracts a great deal of interest as most of the information processed by humans comes from their vision and because despite the efforts, the Human Visual Systems remains weakly modeled.

Many visual algorithms rely on color features. Color histograms computed over transformed colors spaces (HSV, YUV) have been widely used. **L. Agnihotri** [7] proposed quantizing YUV channels at each video frame and grouping them into families based on their similarity, then the selection of most frequent families are used to build an averaged "Superhistogram" which is then used as an index for retrieval or classification. Other examples of use of color features are **C. Lu et al.** [8] which used color signature for classification using Hidden Markov Models or **Z. Rasheed et al.** [9] who used a mix of color features to classify films into genres.

Much effort has been done in the domain of using motion, in 1996 **Ardizzone et al.** [10] proved that motion based features related to the optical flow field could can play a central role in content based video retrieval, albeit features extracted were pretty basic and optical flow computation was costly (back then). The same authors later improved their work [11] by using motion vectors embedded in MPEG streams bypassing the dense optical flow field computation; also the feature extraction evolved to using camera compensated motion vectors histograms, which are calculated at separate quadrants of the frame.

By then, general use of motion did not prove very useful compared to other techniques. However it soon demonstrated to be effective on more specific task. **M. Roach et al.** [12] successfully managed to classify videos into cartoon/non-cartoon. Their algorithm proceeded by creating at each frame a binary map of pixels that are in motion, the sum of these binary maps form a time vector that is derived to obtain second order object motion signal. The spectrum of this signal was used for classification via Gaussian Mixture Models. Another good example of specific classification is **M. Lazarescu et al.** [13] proposal to identify types of football plays (long pass, short pass, kick out, etc…) by using camera motion parameters alone. **A.A. Deshpande et al.** [14] used a set of motion related features like global motion, number of objects in motion, presence of objects at the borders of the video frame or the number of macroblocks moving: these features were selectively used in a decision tree to classify video into static, news, earthquake, commercial, sports or "complex" videos.

Anyway good results have been obtained by dealing with motion alone for general classification and video retrieval. **R. Fablet et al.** [15] used local motion related measurements with a probabilistic causal Gibbs model. Using a metric based on KL divergence on those models after complexity reduction, a similarity measure is obtained. Compared to other contemporary works, their strong point is it can handle a larger range of dynamic scene contents.

Of the many possibilities when processing motion in video, one possibility is focusing on trajectories. These have not been much used, but there are still some relevant examples. **E. Sahouria**

and **A. Zakhor** [16] proposed a motion based video indexing system for street surveillance. Using segmentation and tracking algorithms, trajectories of moving objects including cars, people and bicycles are extracted and represented as two dimensional curves parameterized by time. The features extracted consist in a number of wavelet coefficients calculated upon each trajectory dimension. In [17], **F. Bashir et al.**, segment trajectories using a curvature zero crossing approach combined with a clustering routine. Sub-trajectories are then represented using Principal Component Analysis that significantly reduces data dimensionality. They use several techniques to index and retrieve sub-trajectories, including spectral clustering and string matching by computing edit distance.

Most of the modern clustering and classifying algorithms use combinations of the visual approaches commented here. A good example of multi-modal approach is **A. Basharat et al.** [18] work based on extracting a combination of features from spatio-temporal volumes. First, interest points and their correspondences are established using the SIFT operator. The linked points are used to generate trajectories which are further refined by merging them based on velocity prediction. Similar trajectories in terms of motion similarity and spatial proximity are grouped into clusters. Using the SIFT correspondences and the clustered trajectories, regions are formed and stacked at consecutive frames. This way spatiotemporal volumes are formed. Over these volumes, a set of features including color, texture, motion and SIFT descriptors are extracted. The degree of similarity between the features is computed using Earth Mover's Distance. Two videos to be matched are modeled as a bipartite graph, where volumes are represented by vertices and similarities between them are represented as edge weights, the maximum matching of this graph is used to establish the correspondences between the volumes. The score between each pair of matched volumes is then combined towards the final video matching score.

I.4 Overview

A structural dissimilarity measure based on motion information has been developed. It uses a highly customized block matching engine which forms a high density Vector Motion Field. Using this Vector Motion Field, trajectories are extracted via grouping neighboring motion vectors with similar direction. Trajectories are then filtered and described by 4 sets of signatures which are split at key points to remove outliers and ensure each split component belongs to a specific motion. A novel component similarity metric is used to measure distances between components of different videos. These distances are stored and subsequently processed to extract a final distance between videos using a newly developed double pass algorithm based on the analysis of component distances density estimations. Finally, a hierarchical clustering is carried to check the general validity of the metric. Also three other

video metrics are explained and tested against our proposal. A global block diagram of the whole process can be seen in Fig. 1.

This work is organized as follows: trajectories extraction is described in chapter II; trajectory processing and component distance metric is explained in chapter III; video dissimilarity computation is detailed in chapter IV; clustering results over a video database are presented in chapter V; finally chapter VI concludes this work.

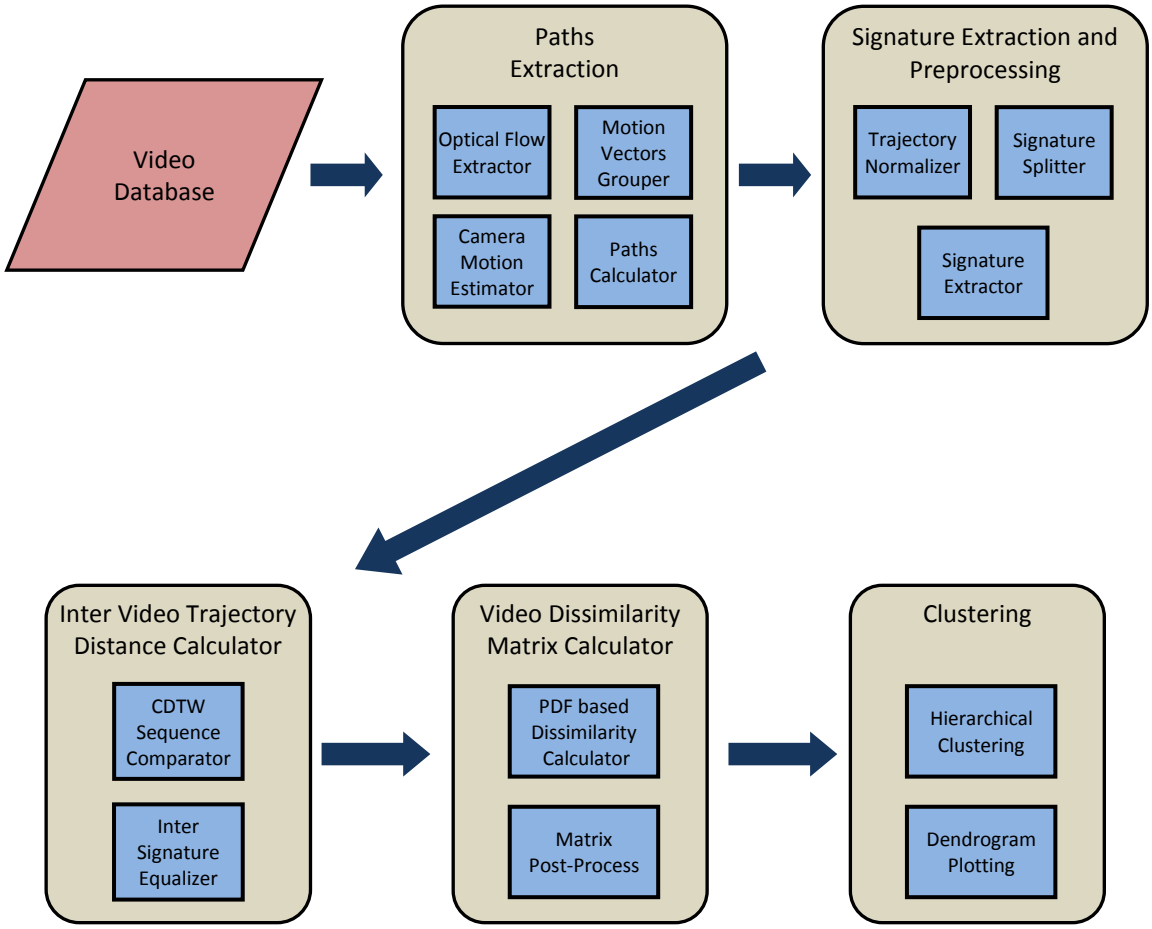


Fig. 1 General Workflow

II Motion extraction and trajectory computation

In this section, the methods for extracting motion and computing trajectories present in the videos are explained. A block matching approach has been chosen because it is easy to configure and customize, it can track large displacements (fast motion) and the resulting Motion Vector Field (MVF) can be used for additional purposes, like easy camera motion estimation.

The basics of optical flow are described as well as the base of block matching algorithms in II.1. Particularities about videos used in this work are detailed in II.2. Modifications in our customized algorithm to identify legit Motion Vectors (MV) and discard/filter noisy ones are clarified in II.3 and II.4. Camera Motion estimation used to correct MVs so relative motion is captured is explained in 0. In II.6 we explain how similar intra-frame MVs are grouped together. Finally in II.7 we describe how these groups are linked between successive frames to form 3D paths.

The workflow consists in 4 processing blocks as shown in Fig. 2:

1. MVF extraction using a custom Block Matching algorithm.
2. Camera Motion estimation using MVs
3. Grouping and labeling similar MV into “Motion Groups”
4. Paths calculations using Motion Groups tracking

For each video, the resulting data is packed and passed to subsequent trajectory processing and analysis blocks explained in chapters III and IV.

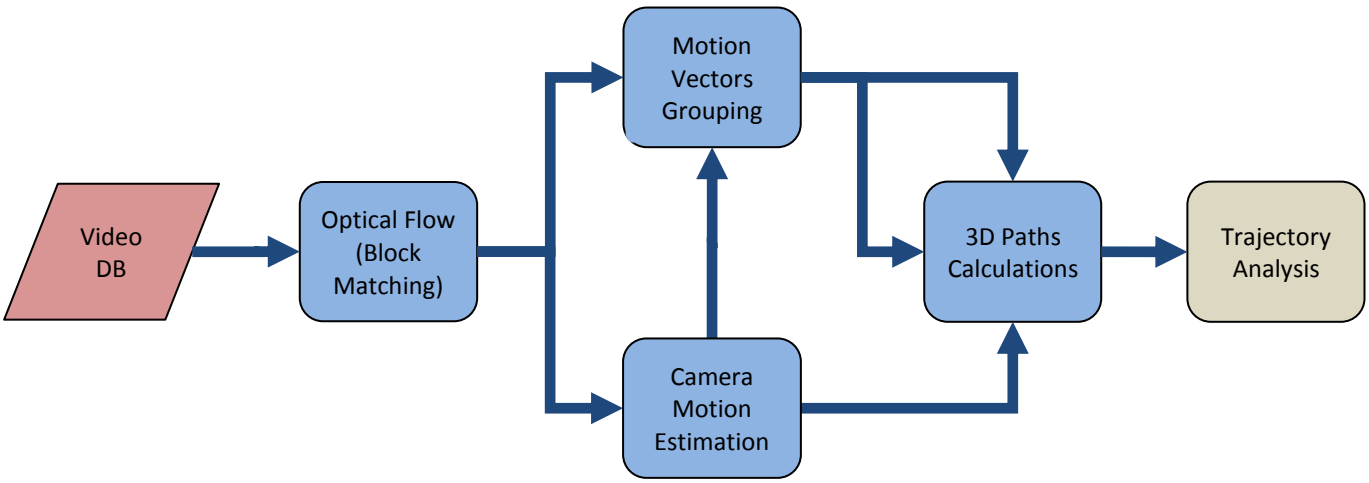


Fig. 2. Motion extraction and trajectory formation workflow

II.1 Optical flow and block matching basics

The main idea behind optical flow is that the future image is a function of the past image that has been displaced locally, hence the future frame can be put as:

$$I(\vec{r}, t) = I\left(\vec{r} - \widehat{D}(\vec{r}), t - \Delta t\right) \quad \text{Eq. 1}$$

This is a rough approximation as it only takes into account basic motion, it does not consider illumination changes, texture variations or new objects appearing, so there is a difference between the future and past translated frame, this is defined as the Displaced Frame Difference (DFD):

$$DFD\left(\vec{r}, \widehat{D}(\vec{r})\right) = I(\vec{r}, t) - I\left(\vec{r} - \widehat{D}(\vec{r}), t - \Delta t\right) \quad \text{Eq. 2}$$

The point is then looking for the Displacement Vectors that minimize the DFD, these define the approximation of optical flow. For this purpose a block matching technique is used, which is probably not the best one when dealing with local low speed motion, but we chose it because it offers a number of advantages:

- straightforward to implement
- can be tweaked easily to match specific purposes
- can cope with long displacements / fast motion (if configured to do so)

The typical Block Matching Algorithm (BMA) segments the frame in a partition of blocks. For each block in a given frame, the position in the previous frame that yields the most similar block is searched. A metric is needed to establish similarity between blocks. Classic options are SSE (Sum of Squared Error), SAD (Sum of Absolute Differences) and SATD (Sum of Absolute Transformed Differences), the last one giving the best similarity in terms of human perception. However as we are not looking for minimum perceived error but actual motion, there is no point is dealing with the increased computational cost, so SAD is chosen as it is the fastest of the group. The minimization to be carried to find the displacement vector (MV) for a given block is defined by Eq. 3:

$$\min_{\Delta x, \Delta y} \sum_{x \epsilon B} \sum_{y \epsilon B} |I(x, y, f) - I(x - \Delta x, y - \Delta y, f - 1)| \quad \text{Eq. 3}$$

Where B is the reference block in the past frame, and Δx and Δy are the coordinates of the displacement vector for that block. In block matching, the displacement vector is better known as Motion Vector, abbreviated as MV.

This minimization implies a full search across the whole image for each block. For a CIF resolution video (352x288 pixels) this means over 10^5 searches for every block, considering a size block of 16x16

pixels this yields over 3.10^{10} operations for each frame, three times more if chroma information is also taken into account. For that reason more efficient suboptimal algorithms are used to conduct those minimizations, based on the premise that the error is a concave function and that the minimum is relatively close to the origin (the displacement between consecutive frames is usually short).

Using a BMA for motion estimation is useful because such information is already incorporated into most compressed video streams and is easily accessible. This information is present for the majority of the frames, and for those that it is not (intra coded frames) it can be interpolated. Unfortunately, basic fast BMAs often used for video compression are not designed to obtain "real motion"; instead they focus on trying to obtain high PSNR while coding the MVs with few bits, no matter what the real motion is. For this reason, in general they are not designed to cope with long spatial displacements between frames (fast motion) as their search parameter is not very big. However, in this work it is important to track even fast motions so it is necessary to sacrifice efficiency for better capabilities and accuracy. Determining how much reduced fidelity BMAs of a given video codec affects final results is so simple as configuring the extraction module described here with the same search parameters of the target codec.

The algorithm used in this work uses a typical search area of 63 pixels, half-pixel resolution and a set of tools to determine whether MVs information should be trusted or not. Sub-pixel motion estimation has been tackled in several ways, two variants can be found at [19] and [20]. Here, for simplicity, each frame is interpolated by a factor of two to use them in the final half-pixel search.

II.2 Video database

Modern video footage uses a wide range of resolutions; 320, 480, 720 and 1080 lines are amongst the most used, also two types of scanning exist: progressive or interlaced.

In this work we used videos at reduced resolution of 352x288 pixels (CIF) and progressive scanning. Reduced resolution has been adopted to make the optical flow extraction faster, also it has the benefit of reducing noise and compression artifacts when videos are down-sampled from higher resolutions. Interlaced videos are avoided because they require separate field matching algorithms and ulterior MV merging.

Videos selected to test our work have been picked from the Internet, more precisely from youtube video hosting website. Video streams have been downloaded with "aTube catcher" software at highest possible resolution and converted to CIF resolution, progressive scan, 2400KB/s bit-rate using the H264 codec included in "Format Factory" software. When imported into Matlab, the videos are converted to Y-Cb-Cr color space. Their length is between 120 and 500 frames each, all of them contain a single shot, hence without scene changes. Seven "families" of videos have been selected to test this work, they are grouped as follows:

- **Weightlifting**: a single person practicing Olympic weightlifting, Powerlifting or Bench Press.
- **Pool diving**: one or two people diving into a swimming pool from a springboard or a platform. Two people are present for synchronized diving.
- **Road surveillance**: vehicle traffic at roads or highways taken from surveillance cameras. Day and night samples are included.
- **Billiard**: breaks of nine-ball games.
- **Soccer**: multiple players playing soccer.
- **Casino**: fixed camera shots of people handling cards or chips, mainly croupiers.
- **Dancing**: groups of people performing choreographed dancing.

All the clips used here have been uploaded to the web. Links to them can be found in the Annex VIII.6.1 , also a link to download the full database is provided.

II.3 Motion Vectors extraction

II.3.1 Image filtering

Prior to the BMA, a configurable filtering stage is applied. Fast block matching algorithms rely on supposition that the cost function for blocks comparison is concave, which is an assumption that is in sometimes false. The purpose of filtering is smoothing the images to reduce the probability of the algorithm of being stuck in local minima, also it effectively reduces the effects of noise and video compression artifacts. Several filtering options have been tested:

- **Linear filtering**, using a 5x5 gaussian filter with $\sigma = 1$.
- **Wiener adaptive noise filtering**, with a 3x3 estimation area and noise modeling for AWGN.
- **Anisotropic diffusion filtering** with 2 iterations and a gradient threshold of 15.

Examples of these are shown in Fig. 3. They have been tested against noise and JPEG type compression artifacts. JPEG compression has been selected for testing because it uses DCT coefficients to code the image, just like most video codecs do.

Linear filtering is the least effective method, as for equivalent results the smoothing effect is much higher, rendering images too soft. Its only advantage is it is very fast to compute. Anisotropic filtering excels at processing JPEG artifacts, almost restoring the image to its original state (except for some details lost by compression). This method has the ability of preserving edges very well. It also removes noise pretty well; but when SNR is too low, originally smooth textured surfaces are cluttered by “salt and pepper” noise. Wiener adaptive filtering has an overall good behavior, it reduces a great deal noise with few detail loss and it also works well with compression artifacts. However, it does not preserve edges as well as anisotropic filtering. Finally, it is computationally faster than anisotropic filtering.

The optimal filter selection depends on the video source. For uncompressed, daylight videos, it is better just not filtering at all, as more high frequency textures are preserved. When bad quality cameras have been used (webcams, mobile phones) or harsh conditions happened at filming (dim light), Wiener filtering is preferable because it deals noise better without risk of producing “salt and pepper” noise. Finally if original footage is of good quality, but it has been compressed in excess, anisotropic filtering is the better choice.

The videos used here belong to the last category. No significant digital sensor noise is visible, most of the videos have been taken with professional camcorders, and when it is not the case, filming has been carried in broad daylight. On the other side, they have been downloaded from video hosting websites, sometimes with relative high compression rates. Because of this anisotropic filtering has been selected as the default filter.

Fig. 3. Image Filtering. Cropped examples of the three filtering techniques considered. Left row shows the results for noise added image (AWGN with 20dB SNR). Right row shows results for JPEG high compression (Q=25/100).

II.3.2 Macroblock size and spacing

In video compression codecs, macroblocks form a partition of the image. Each pixel movement is hence determined by a single macroblock. This is efficient in terms of video coding, but it is not good enough for optical flow estimation as the resolution resulting of the Motion Vector Field (MVF) is low: for a CIF resolution video the MVF yields only 22x18 MVs. In this work we use macroblock overlapping, distances between these has been reduced from the standard 16 pixels to only 4, thus increasing density by 4. The counterpart is an increase in computational cost by 16, thus slowing dramatically the speed of the algorithm.

Macroblock size determines how many pixels are compared when determining each MV. The bigger the macroblock is, the less prone it is to falling into local minima, but the less efficient it is at locating small objects movements. MPEG1 uses 16x16 pixels blocks, in our work it is a configurable parameter, but 12x12 pixels has been the selected default value.

II.3.3 Cost function

The cost function to be minimized takes into account all three video channels. Intensity channel is usually good enough for this purpose, but adding chrominance channels adds robustness in some situations, especially when objects of similar luminance and smooth texture get close.

Aside from modifying macroblock size to adjust the sensitivity to detect smart objects' motion, it is useful to adjust the weight of central pixels compared to the outer ones. To do such, a window is applied to the absolute difference for each pixel. This window is selectable between:

- MB size/2.5 pixel standard deviation – "Thin" raised Gaussian
- MB size/5 pixel standard deviation –"Thick" raised Gaussian
- No windowing

This parameter is better chosen depending on the video content, however the thin raised Gaussian window has been the preferred one in this work.

The cost function of the (i,j) block for luminance channel can be written as in Eq. 4:

$$C_{Y\,ij}(\Delta x, \Delta y) = \sum_{x=i}^{i+\mathrm{MB}} \sum_{y=j}^{j+\mathrm{MB}} H(x,y) \cdot |Y(x,y,f) - Y(x-\Delta x, y-\Delta y, f-1)| \quad \text{Eq. 4}$$

Where H is the window, Y is the luminance channel and MB is the macroblock size.

The equivalent cost function is computed for Cr and Cb channels, and then all three are combined as follows:

$$C_{ij} = \alpha C_{Y_{ij}} + \beta C_{R\,ij} + \gamma C_{B\,ij} \quad \text{Eq. 5}$$

Experimentally luminosity has shown to have more energy than Cr and Cb channels, on the other hand Cr and Cb components use to be sub-sampled in most videos (4:2:2 with our encoder, but the original source may have even higher chroma sub-sampling). Finally, $\alpha = 1$; $\beta = 0.7$; $\gamma = 0.7$ weighting factors have been chosen.

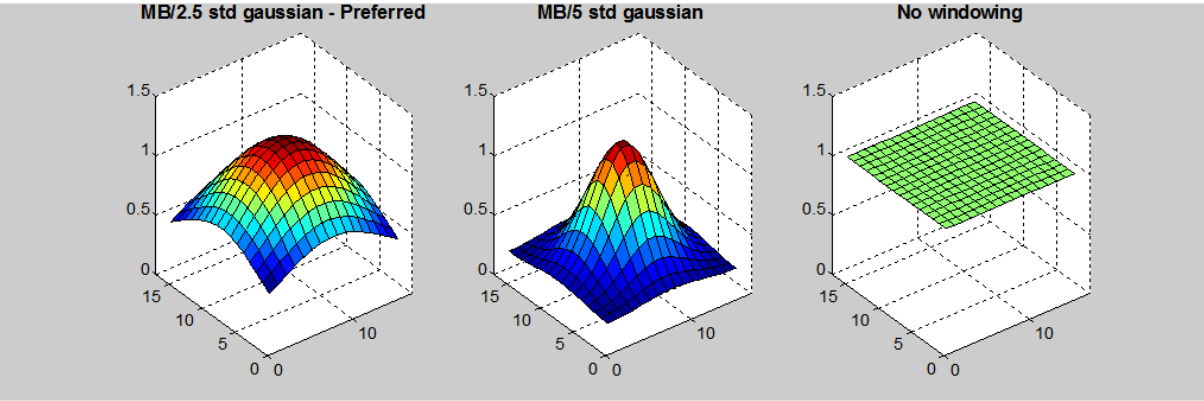


Fig. 4. Cost windows

II.3.4 Search grid density

A typical N-step algorithm like in [21] would start looking through a 3x3 grid and halve the search step size at each iteration, the problem here is that to cover a 62x62 search area the first step size would have to be of 16 pixels (with 8 pixels grid spacing the search area would be of only 46x46). This is too much and there would be a great chance of getting stuck at local minima. Also other fast algorithms like 2D-log could choose a false 1st step leading to a wrong result.

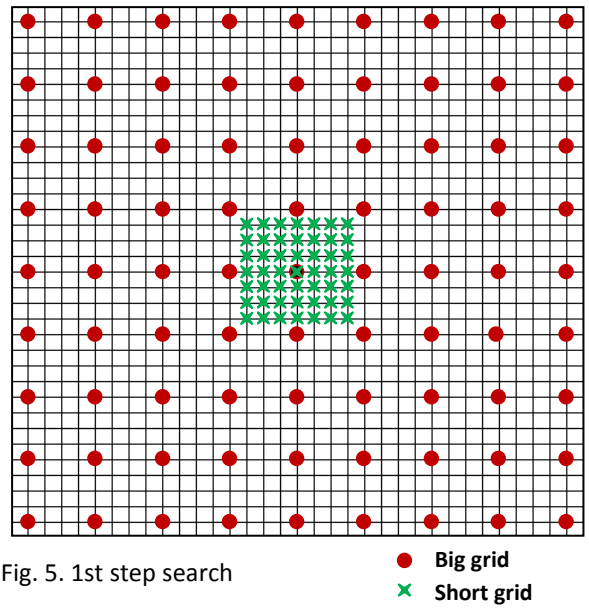


Fig. 5. 1st step search

For this reason speed has been sacrificed again for higher precision, in this approach denser and bigger grids are computed in two steps for 1 pixel precision, with a third step for half-pixel precision refinement.

II.3.5 Search steps

In the first step, two grids are computed. The larger one takes 9x9 searches with 4 pixel spacing, the short one takes 7x7 searches with 1 pixel spacing, an illustration can be seen in Fig. 5. From all the results obtained, the minimum is selected and the MV for that block is updated accordingly. If the minimum is found at the large grid, the second step is executed, if it is found in the short one the search proceeds directly to the third step as 1 pixel precision has already been reached. The double grid search is performed because most times the motion falls close to the center, in this case there are lower chances of getting wrong results because of similar regions across the search area, also no computation penalty is added if the motion is short.

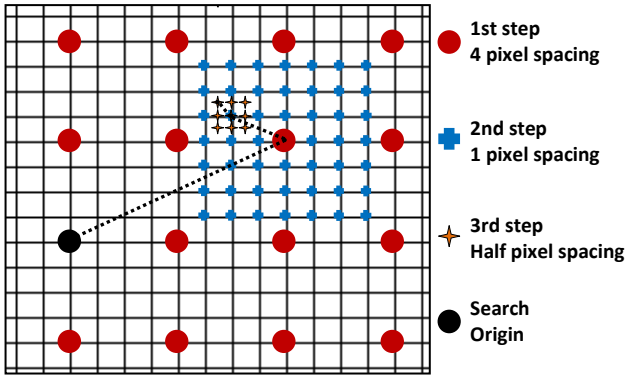


Fig. 6. Three steps example search (detail)

If the second step is carried on, the search center is updated to the result location of the first step and a new 7x7 search grid is computed. At this point we have the MV at 1 pixel precision.

The third step uses cubic interpolated frames to calculate a 3x3 sub-pixels grid, which increases resolution to the half-pixel level.

Note: Interpolated frames are computed using the actual video frame, without image filtering, because maximum high frequency detail is necessary for that step. This means that low quality videos do not really benefit of the half-pixel precision, because will often give false motion estimation. Anyway the loss of motion information at this point is not important.

II.3.6 Ambiguous macroblock discarding

Under some circumstances there is a high probability of erroneous motion estimation:

- At the borders of the picture, motion cannot be calculated for objects moving in or out the frame.
- Under presence of two objects with smooth texture and with a rectilinear boundary, the algorithm is easily fooled (aperture problem).
- At luminance clipping areas or where texture is too smooth, the cost function returns very low values for all search positions.

The first problem is avoided by discarding outermost macroblocks. For the other issues a previous stage determines whether there is enough detail information to allow for confident motion estimation. Edge detection with a Sobel filter is used: if no edges are present in a given macroblock this one is

tagged as ambiguous. This way a mask with the same size as the MVF matrix is computed and used latter in subsequent stages. An example of this MV mask based on edge detection is provided at Fig. 7.

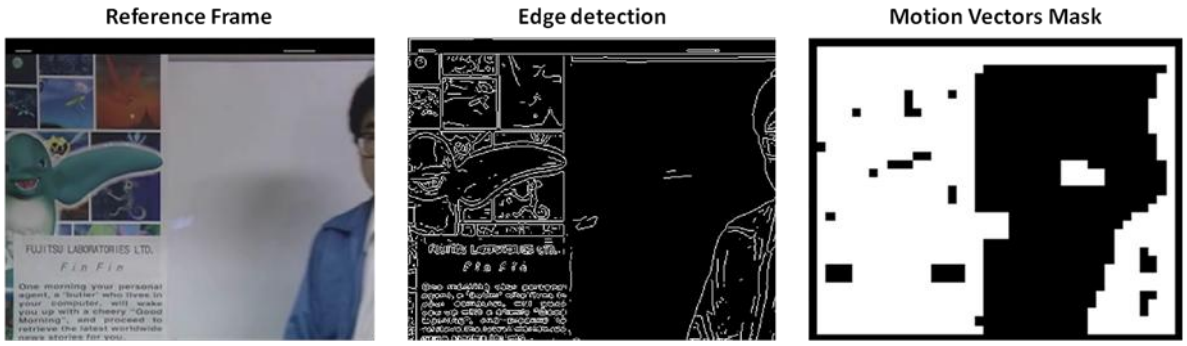


Fig. 7. Edge detection for block discarding

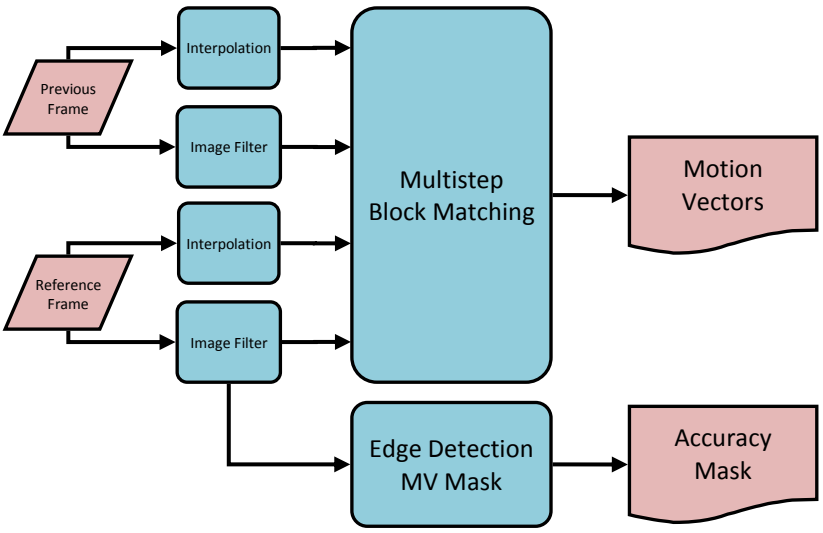


Fig 8. Motion Vectors and Accuracy Mask extraction workflow

II.3.7 Cinematic band detection

Not all the videos use the same aspect ratio. Here, we are working with a 4:3 ratio, however 3:2, 16:9 and others are also usual. When encoding a video, if input and output aspect ratios differ, possible solutions are cutting, expanding/shrinking or adding black bands to the image. The first two solutions are not a big deal for this motion extraction implementation, aside from eliminating information or slightly deforming the image (and hence, extracted trajectories). However, adding black bands creates a big problem for the BMA.

The presence of a black surface with rectilinear shape fools the BMA. If the reference macroblock which motion is being estimated includes part of the black surface, the resulting cost function will

privilege positions close to that black border. If luckily there is no motion at this position, the MV will be correctly calculated as zero motion, but if motion is actually present in any direction different from the black surface edge (horizontal in the case of cinematic black bands), the MV will be erroneously estimated. The result is “false” MVs along the black bar’s edges, in the end this translates into many false short and noisy trajectories.

To avoid this, black band detection has been implemented. Black bands have two main properties which are exploited here: they use the “absolute black” for the video (which does not have to be 0, usually it is 16) and they are placed horizontally.

In our algorithm, the median along horizontal lines of the video is calculated. This is, using the luminance channel, for each horizontal line of the video the median is computed. This median is not calculated for a single image, instead it is carried along all the frames. The output is a vector of length equal to the vertical resolution of the video, in our case 288 pixels. We call this vector the luminance profile of the video. Ideally the black bands would stay at “absolute black” level, however this is not the case: lossy video compression spreads residual errors in the edge vicinity, so threshold based detection is needed.

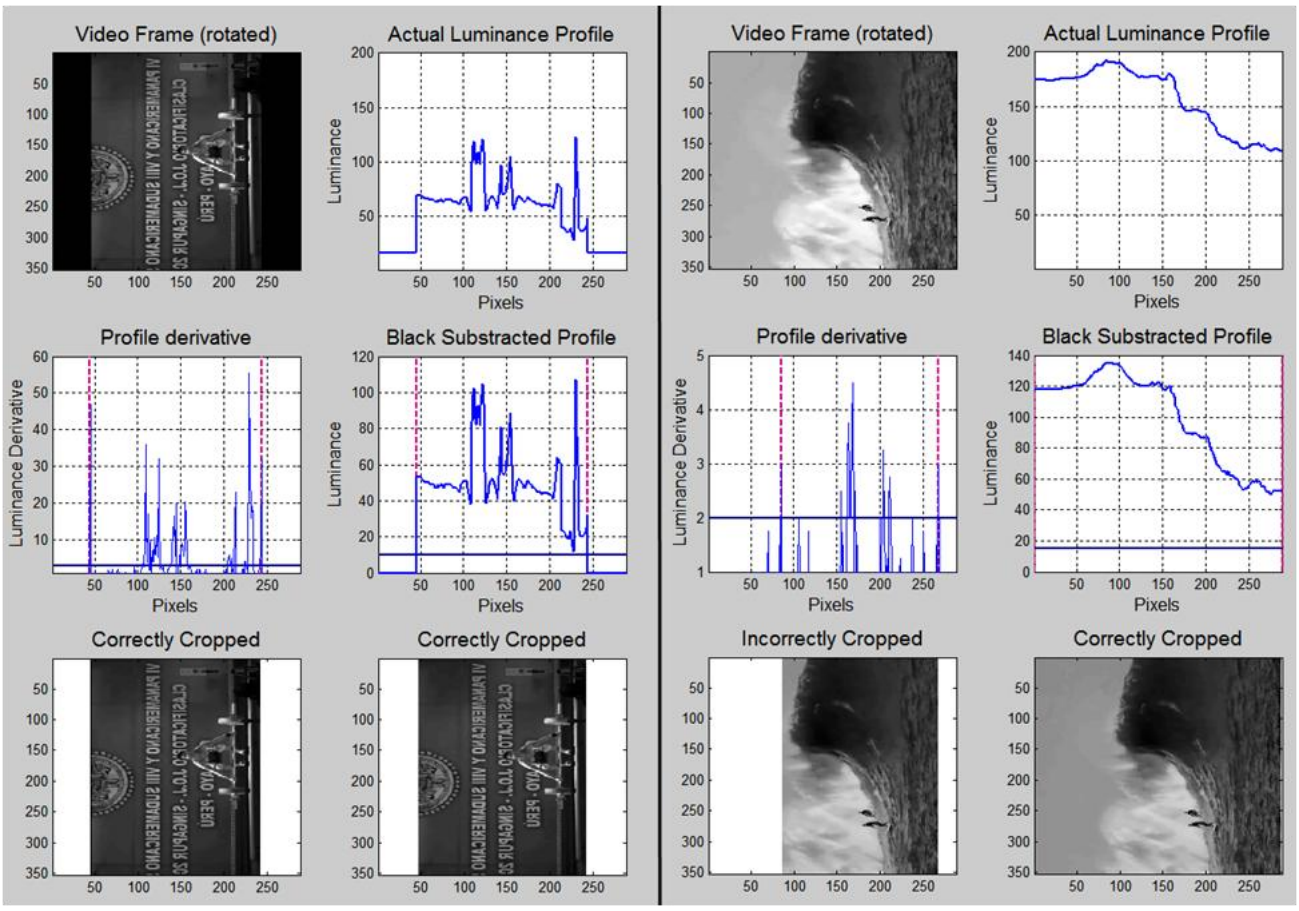


Fig. 9. Two Band detection examples. On the left, a video which actually has black bands, on the right one which does not. Derivative profile gives a false positive on the second one because of the smoothness in the frame borders vicinity.

One possible approach is calculating the profile's derivative, then search for the first and last points where it crosses a given detection threshold. Another solution is applying directly a threshold after subtracting the minimum luminosity level present in the video to the luminance profile. Both solutions work very well at detecting the band's positions, but the first one sometimes fails when the video has no bands at all. This flaw happens when a video without those bands has a smooth texture on its upper or lower limits. For example, a video with only a clear sky on the top would detect the upper band edge far from the actual frame border, this would not happen with the second method, which is the selected one in this work. Two examples with both methods are shown in Fig. 9.

II.3.8 Frame discarding

Video conversion sometimes leads to frame rate adjustments. Encoders deal with this in several ways, some of which cause problems to the motion extraction method used here.

In some cases, the adjustment is done by simply eliminating or replicating a frame. Elimination is not a big deal because as it will be explained in section III, the trajectories are filtered. However duplicating a frame causes all the trajectories to be broken at that frame as the MVs between the original and duplicated frame are all zero. To solve this issue conditional frame skipping can be implemented easily: if the MV matrix for that pair of frames is zero, the frame and its associated MVF are discarded. A more efficient way to implement this is calculating the frame to frame absolute difference, which is zero when the next frame is a copy of the previous one.

Other encoders use more sophisticated techniques that involve motion interpolation. When this interpolation is of good quality, the resulting video is correctly processed by our motion extraction software. Unfortunately this is not always the case, sometimes the interpolation process tends to generate frames which are very similar to their predecessor, but not equal. In that event, as the frame to frame difference is not zero, the simple conditional frame skipping does not detect a "repeated frame" situation, but at the same time the motion extraction routine fails to compute MVs adequately, subsequently causing trajectory breaks.

The adopted solution is computing the frame to frame difference and skipping them when it falls below a given threshold. However, the difficulty is fixing that threshold as a high value would give false positives in videos which motion is caused by few little objects moving over a static background.

A conservative value has been chosen after examination of a large enough set of videos, the reason is spotting false negatives is easier than detecting false positives that would distort trajectories. In case of false negatives the video threshold can be readjusted and the video processed again.

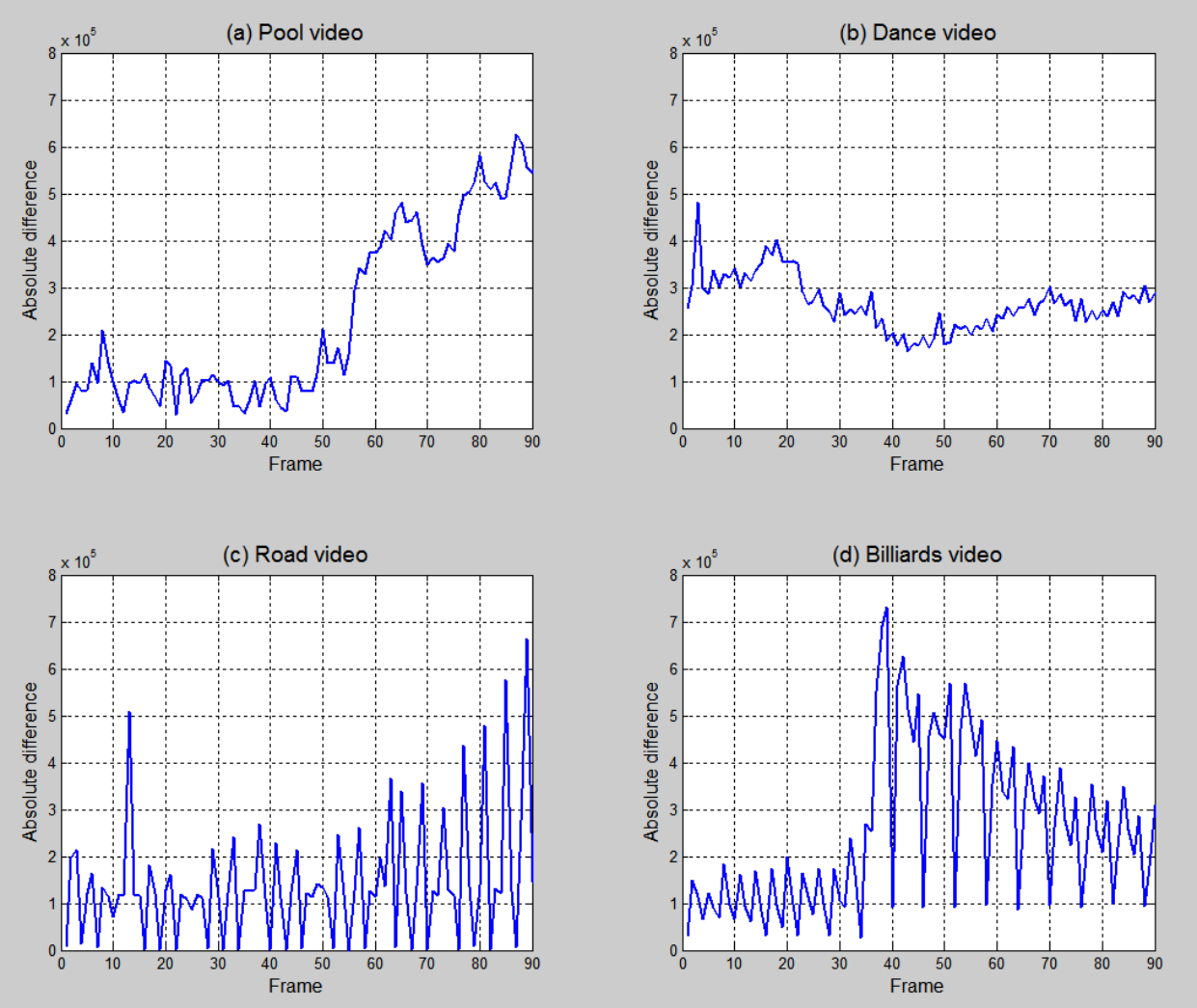


Fig. 10. Frame to frame differences examples. (a) and (b) show correct behavior. (c) shows the effect of frame repeating encoding. (d) shows the effect of "defective" frame interpolation.

Fig. 10 shows four examples of absolute frame difference. The "pool" and "dance" videos have standard behavior, the frame difference varies along time depending on the scene change, but it is relatively continuous. On the other side, "Road" and "Billiards" show problems at this respect. More precisely "Road" is an example of video where frames have been copied, this translates into zeros in its frame difference vector. "Billiards" video exhibits some kind of frame interpolation, here the difference is never zero, but it is highly discontinuous; furthermore it is a good example why setting a threshold is difficult, as the "legit" differences in the first half of the video (peaks) are close to the differences caused by interpolation of the second half (minima).

II.3.9 Examples of MV extraction.

Below can be found two links to animations that illustrate the extraction of MVs. Video and MVs are superimposed for comparison between actual motion and estimation. All the links to videos used in this report can be found in the annex.

Bowing MV extraction: http://www.youtube.com/watch?v=MTFzznP6IjQ&hd=1

Synthetic MV extraction: http://www.youtube.com/watch?v=J43q19pQxHo&hd=1

II.4 Motion Vectors filtering

MV estimation is not always accurate, especially under harsh conditions like high noise/compression, overlapping objects or fast and blurry motion. Here we take advantage of the high density MVF by filtering MVs, thus obtaining a smoother flow estimation.

Erroneous MVs do not follow a classic noise pattern distributed along all the MVs, instead they behave more like outliers completely uncorrelated from their neighboring MVs. Under this circumstance linear filtering is a poor choice, it is better to apply median filtering to discard the information of such MVs, this way noisy MV are corrected without affecting its neighbors. The size of the median filter has been chosen is 3x3, a larger filter would result in better outlier removal, but it would compromise final MVF resolution because of the increased smoothing. Fig. 11 shows an example of MVF filtering from the “Bowing” sequence.

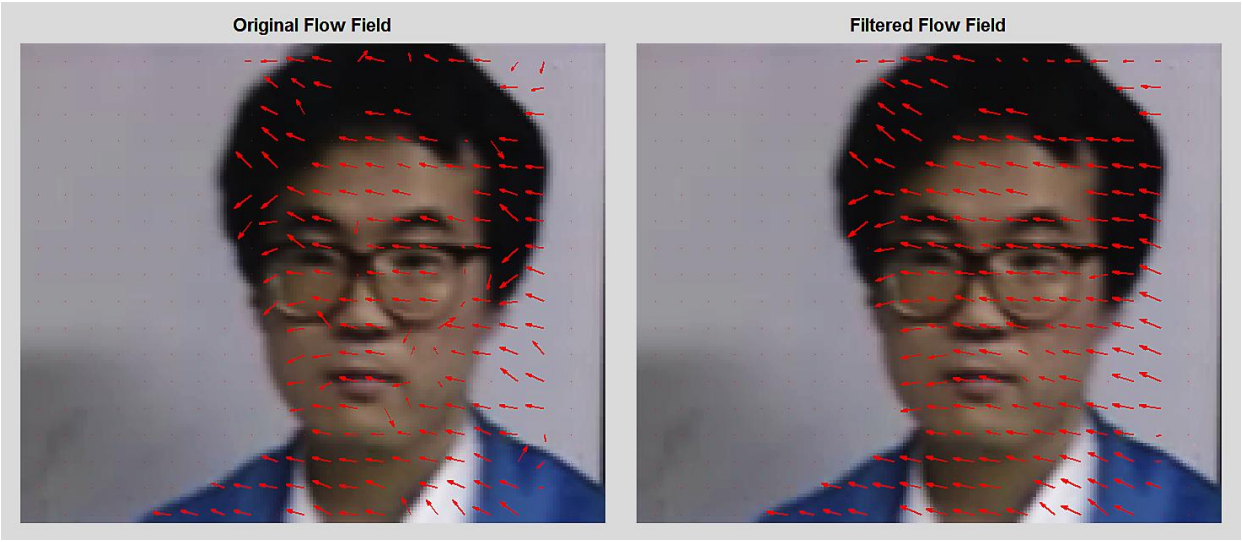


Fig. 11. Optical flow field filtering

II.5 Camera Motion estimation

Determining camera motion is essential to calculate trajectories relative to the background, because perceived motion depends on the movement of objects respect to their background. Sophisticated camera motion estimation algorithms can be found in the literature, for example **Yeping Su et al.** [22] used MVs embedded in MPEG2/MPEG4 video streams to fit a parametric perspective motion model (8 parameters). Such methods use to be computer intensive and even so they are prone to inexact estimations when big enough objects move across the frame.

Advanced camera motion is out of the scope of this project, however basic camera panning can be very useful for extracting relative motion. Under camera motion the directly computed paths are very different from the real world paths the objects follow relative to the background. If the camera tracks a main object moving across a scene, as the object itself does not move inside the frame it would not generate any trajectories (or just some short and noisy ones) while the displacing background would be estimated as a huge object and it would generate a trajectory with the shape of the camera motion.

The approach used here only considers a simple translational model. Panning is the most usual camera motion followed by zooming, and is easy to compute using MVs. One straightforward way of obtaining translational camera motion is just taking the median MV from the MVF. If moving objects do not cover a large area of the frame, this method is just good enough. However when a large portion of the frame is cluttered with moving objects the median vector may come from the motion of a non background object.

Background MVs have two relevant specificities that are exploited in our algorithm: they are in large numbers and they have little variance (they all share the same direction and speed, except for some little jitter caused by the block matching stage).

Our camera motion estimation uses 2D kmeans partitions on the MVF at each frame, and identifies the partition which contains the MVs generated by the moving background. This is done multiple times with a different number of initial bins (3 to 8). At each iteration a likelihood function is calculated for each resulting bin, the maximum is taken as the candidate bin for background motion. The likelihood function is:

$$L = \frac{n}{sumD + \alpha} \qquad \text{Eq. 6}$$

Where n is the number of elements in the cluster, sumD is the within-cluster sums of point-to-centroid distances and α is a small constant to avoid instabilities. This function takes into account the number of elements, but also how similar they are as sumD decreases when the MVs from the

computed bin get closer. The cluster with higher likelihood is selected and its median MV is taken as candidate for camera motion.

Once this has been done for every initial number of bins, the mode is taken as camera motion. In case there is a tie between two or more candidates, the algorithm overrides to the simpler median of all motion vectors in the frame.

Experimentally about 80% of the time all iterations yield the same result (the mode for all iteration was the same, only in about 3% this technique is inconclusive, but even then the selection of simple median may return actual camera motion.

Note: when selecting motion candidates from the higher likelihood cluster, median is used instead of the centroid, this is because at different iterations the centroids corresponding to the "equivalent" bin do not use to be exactly the same, while median does. This is because of the discrete nature or the MVs: in a discrete set, the addition of a close but slightly different MV to a cluster will inevitably displace the centroid of that cluster, but the median will remain the same while there is a high number or MVs with the same speed and direction. Hence if we use the centroid as candidate, probably all the candidates will have different value and their mode will be one. Fig. 13 shows an example of three iteration clusterings.

This technique has some limitations:

- Under very fast camera motion, the image is blurry and hence MVs are not accurate enough.
- With zooming, the background MVs no longer share direction and speed, compromising effectiveness. Anyway mild zooming is acceptable as the variance of the MVs corresponding to the background will not reduce the likelihood factor too much.
- When a very big solid object moves across the frame its MVs may be selected as camera motion if its surface is bigger than the background.

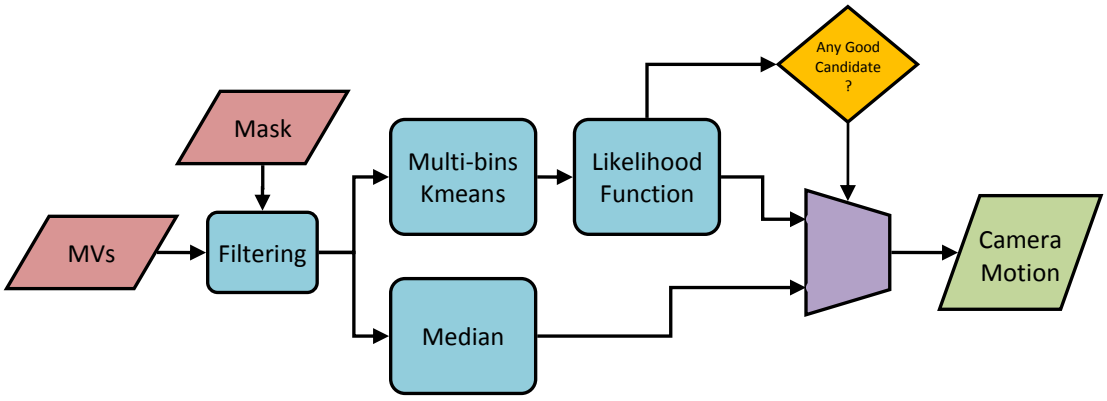


Fig. 12. Camera Motion Estimation

Camera motion information is used to compute compensated MVs and a background mask: camera motion is subtracted from the original MVs to calculate the compensated MVs, then for every compensated MV where motion is below a given threshold (1 pixel typically) the location is considered as background. From now on, when referring to MVs we are in fact talking about global motion compensated MVs.

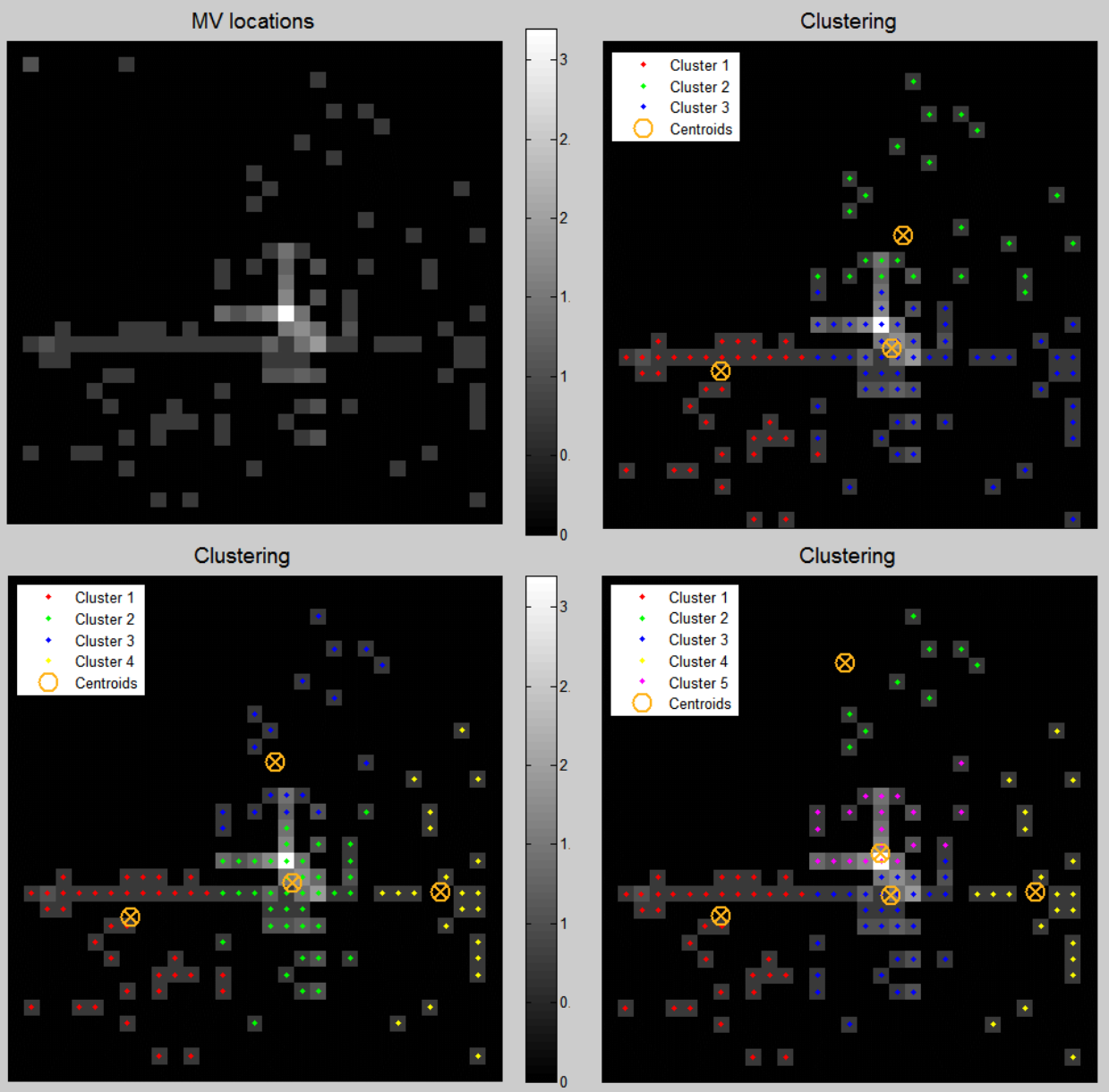


Fig. 13. MVs clustering for camera motion estimation. Pixel luminance indicates the number of MVs with such coordinates in a logarithmic scale. Note: in practice the median of each bin is taken as candidate, not the centroid.

II.6 Motion Vectors grouping

The next step after computing MVs is grouping them at each frame with a given criterion based on the angle of their motion and their connectivity. Basically, for a given reference MV, all MVs in its vicinity that meet the similarity criterion are grouped and labeled together. An iterative algorithm is used for this purpose, starting from the first group until it is not possible to form groups over a threshold group size. Note: the background mask from the previous stage is applied: MVs considered background are not taken into account.

First a neighborhood affinity matrix is calculated: using the metric below, for each MV, the "Neighbor Affinity Factor" (NAF) measures how much the surrounding MVs are similar in motion to it. The idea is finding the MVs which are likely to be the most representative for each motion group. This approach for selecting reference MVs results in more accurate frame segmentation than proceeding by selecting sequentially every still ungrouped MV as reference for the next group.

The NAF for the (m,n) MV is defined as follows:

$$NAF(m,n) = \sum_{i \epsilon S} \sum_{j \epsilon S} \frac{\alpha_{max} - \min(\Delta\alpha; \alpha_{max})}{\alpha_{max}} \left[1 + \min\left(\frac{v_{ij}}{v_{mn}}; \frac{v_{mn}}{v_{ij}} \right) \right] \quad \text{Eq. 7}$$

Where α_{max} is the angle difference threshold (typically 30º), $\Delta\alpha$ is the angle difference between the reference (m,n) and (i,j) MVs, v_{ij} and v_{mn} are the modulus of the respective MVs, and S is the region where affinity is considered (typically a 7x7 area). This function takes into account both direction and relative speed differences, but focusing on the first one.

The MV with higher NAF value is selected as motion group reference. Then, all the MVs in the frame that match the grouping criterion are marked as potential group members. Using a region growing algorithm only those that form a connected region with the reference MV are labeled together as a group. More specifically, a binary image is formed with the positions of the candidate MVs set to 1, using this as a mask and the position of the reference MV as a marker, a morphological reconstruction is conducted, the result is a map of the positions of the connected MVs that share similar motion.

The MVs being assigned are discarded in subsequent iterations, this process is repeated until it is not possible to form groups with a minimum size. Fig. 14 shows a flowchart of the algorithm.

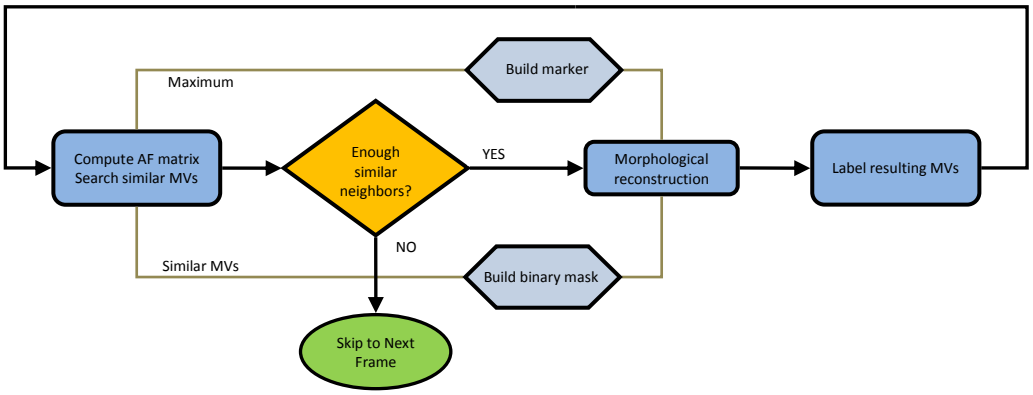


Fig. 14. MV grouping flowchart

II.7 Paths tracking

To track paths, groups at every frame must be compared to those of the previous frame in order to link them when applicable. If there is a good match, both groups are considered part of the same path, if not, a new path is created. For this purpose all the groups from a frame need to be confronted to those of the previous frame.

For each group, key components are calculated, these are:

- **Position**: average geometric center of the MVs
- **Motion Flow**: average of the MVs
- **Size**: number of MVs

This data is used to calculate the Cost Matrix for groups linking, more precisely:

- Distance between present frame groups and predicted position from groups in past frame (prediction position is computed as the past position plus the average motion flow of the MVs of that group)
- Motion Flow difference
- Size difference

The Cost Matrix is then defined:

$$CM(i,j) = \alpha \left| \vec{r}_{G_i} - \left(\vec{r}_{G_j} + \vec{f}_{G_j} \right) \right|^2 + \beta \left| \vec{f}_{G_i} - \vec{f}_{G_j} \right|^2 + \gamma \left| s_{G_i} - s_{G_j} \right| \quad \text{Eq. 8}$$

Where $\vec{r}_{G_i}$, $\vec{f}_{G_i}$, s_{G_i} are geometric center, motion flow and size of the group i respectively. Index i stands for present frame groups and index j stands for past frame groups. α, β and γ are weight constants, 10, 50 and 5 values have been used respectively in this work, these have been chosen by empirical testing.

Group linking is not trivial as paths may end, start or continue, so setting a simple decision rule is not obvious, also the constants must be chosen carefully. A flowchart of this algorithm is provided in Fig. 15.

For each past frame group, the minimum cost association with present groups is searched. If this minimum falls below a given threshold, then the groups are considered to be from the same path, the present group costs from the Cost Matrix are then eliminated so it cannot be assigned to other past groups. If the cost is over the maximum cost threshold, then the path that the past group belongs to is terminated. The chosen threshold value is 1000.

Once this past to present assignation process is over, the algorithm checks for present frame groups that did not get linked and assigns new paths to them.

Two examples of paths extraction are shown in Fig. 16.

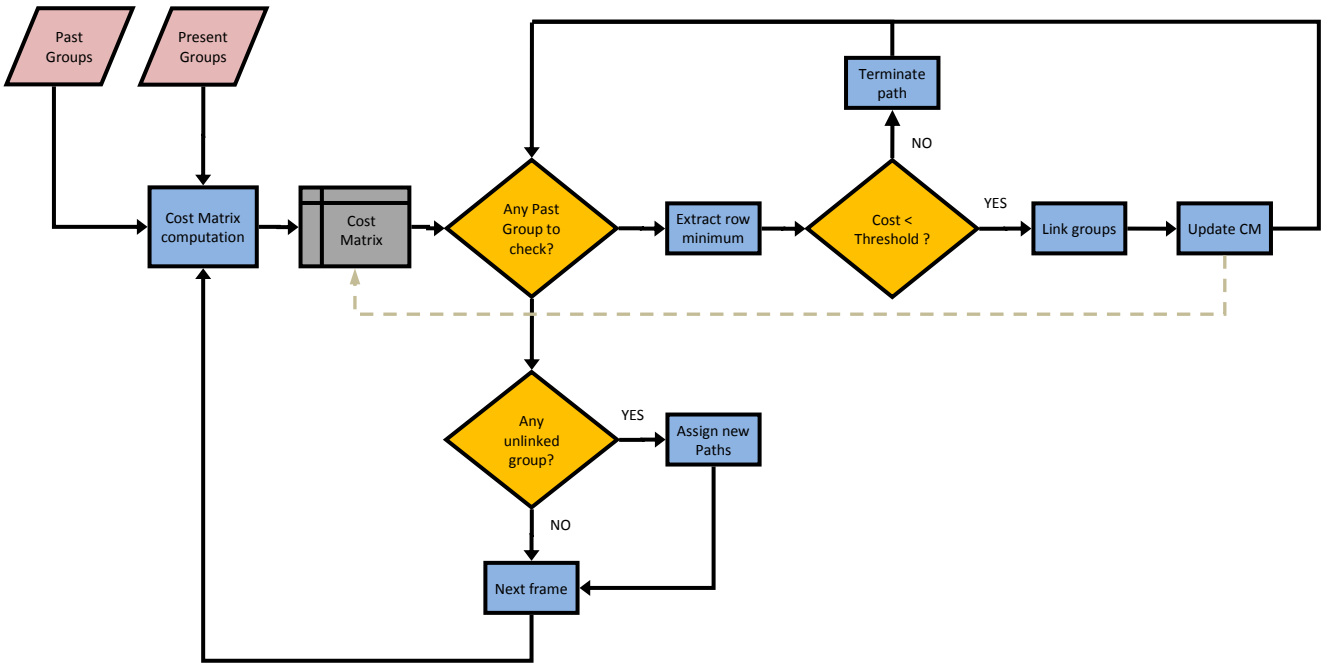


Fig. 15. Paths tracking flowchart

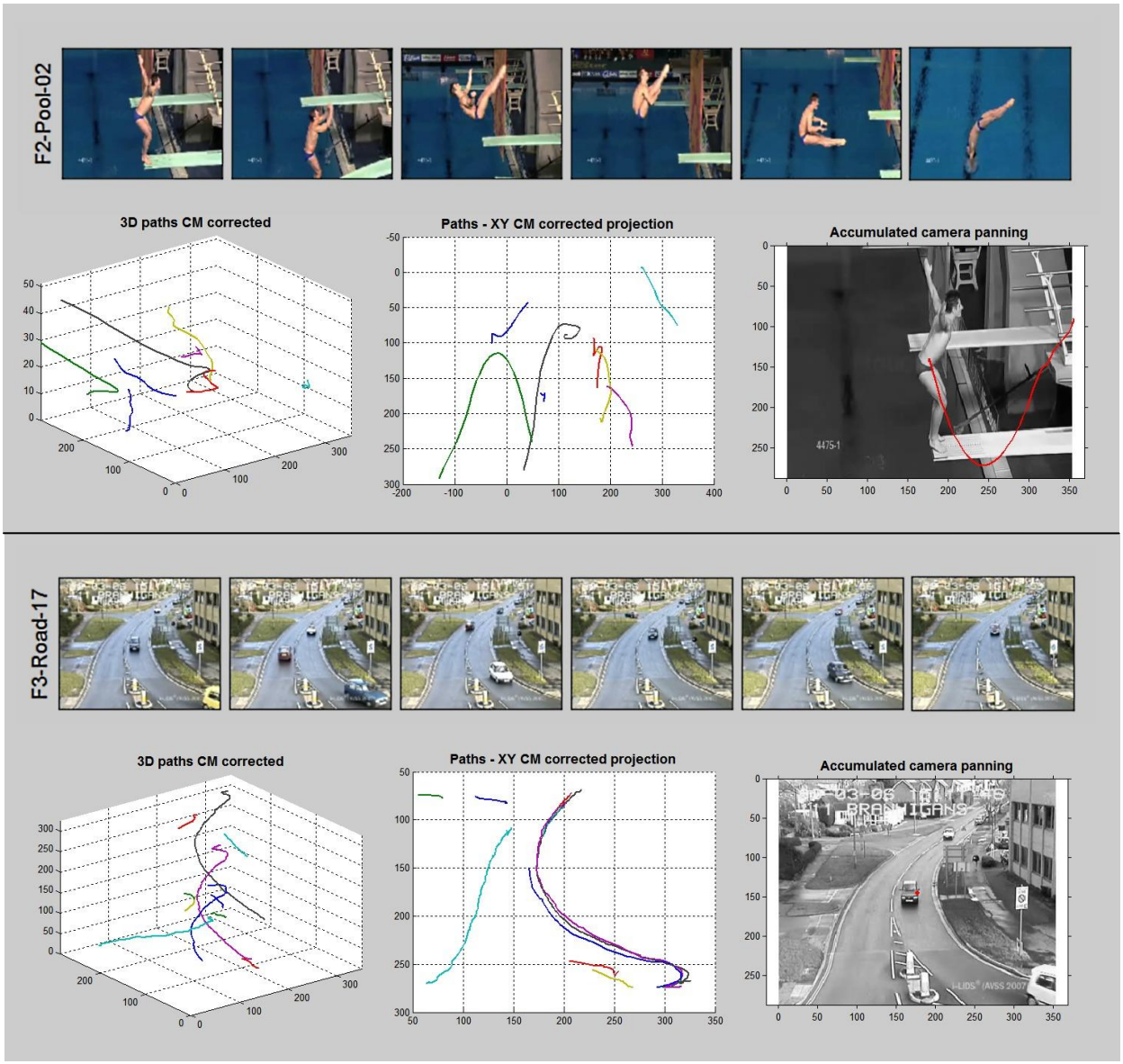


Fig. 16. Paths extraction examples

III Trajectory signatures – Component distance metric

Comparing paths directly is a very difficult task, there is no direct norm that effectively gives a good measure on paths similarity. Two 3D paths which may be considered similar may have different spatial positions, orientation, scale and could be generated at different time; because of this Euclidean distance is almost useless on raw paths.

Historically different strategies have been used to represent trajectories. A straightforward model consist in the use of chain codes [23] [24], which is a string based representation that assigns symbols to key points of the trajectory. For example, **Z. Dogan** [24] used trajectory direction to map a 9 symbol alphabet and compare the resulting strings using a Levenshtein metric [25], thus providing invariance to spatial shift and scaling, also it makes easy finding sub-trajectories via LCSS algorithms. Other methods rely on trajectory simplification like piecewise linear approximations [26] or splines [27]. Principal Component Analysis [28] has been used on raw data as well [29], or combined with previous trajectory pre-processing; PCA is based on the analysis of samples covariance and gets rid of features that do not have significant information, thus reducing data dimensionality.

In this project we decided to convert 3D paths intro trajectories, these being the projections of 3D paths onto the XY plane. Then, a set of primitives based on geometrical features are extracted from the trajectories, each 2D trajectory is divided into four 1D primitives: speed, acceleration, curvature and curvature variation. These four primitives are what we call trajectories signatures. Prior to that, trajectories are smoothed to reduce the propagation of noise into the primitives. Components are then split, like in [30], but cutting signatures independently. To achieve spatio-temporal invariance, a normalization stage is carried at component level. A model for primitives matching is proposed that is used to confront videos trajectories and establish a similarity measure.

In this chapter we first explain how we filtered trajectories in III.1. Then we expound how to compute their signatures in III.2. The technique used to split signature primitives at key points is described in III.3. In III.4, we explain how it is possible to make the global algorithm invariant to space, scale and time by using specific normalizations. Finally in III.5 we detail how we built a metric to measure inter signature components distances, which is the data used by the subsequent block of our global algorithm explained in chapter IV.

III.1 Trajectory filtering

As seen in section II.7, we are using group centroids at every frame to compute paths, this generates a great amount of noise which may affect the trajectories signatures. The cause is mainly that between two consecutive frames, the group size may vary by the addition or subtraction of some MVs, thus displacing the centroid by some pixels abruptly.

There are some characteristics for the filtering stage that have to be taken into account. As some paths are relatively short not any kind of filter is useful, also to keep shapes it is essential that the filters have linear or quasi linear-phase.

To correct phase shifts a forward-backward filtering has been used [31], this technique filters the sequence once (forward direction), then flips the result and filters it again (backward direction). The purpose is getting a zero-phase filtering, this way shapes and geometrical locations are maintained. Note that with such method the order of the equivalent filter is doubled.

We tried many filter types, but as we wanted to ensure final zero phase, we stick to a simple FIR design (these have perfectly linear phase), despite not being between the best in terms of frequency selectivity. The minimum signal length for such filtering is $l = ncoeffs \times 3 - 2$. When possible, a 5 coefficients filter is used (this equals an 8 order at double pass), if trajectory has not enough elements the number of coefficients is reduced until filtering is possible, in that case we chose to keep cutoff frequency in detriment of selectivity. Anyway a lower cap for trajectory length is set to 7 elements, which corresponds to a 3 coefficients filter (and hence 4 order equivalent filtering). Fig. 17 shows an example of trajectory filtering, Fig. 18 shows module and phase response of the selected filter. Coefficients of the selected FIR filter are: a = 1, b = [1 0.55 0.3 0.12 0.05].

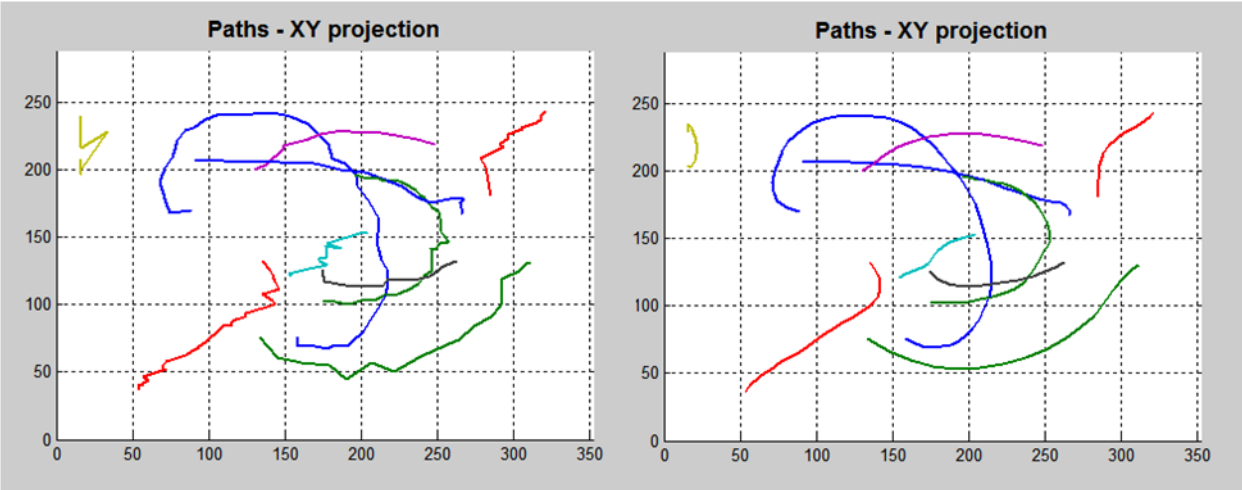


Fig. 17. Path filtering example

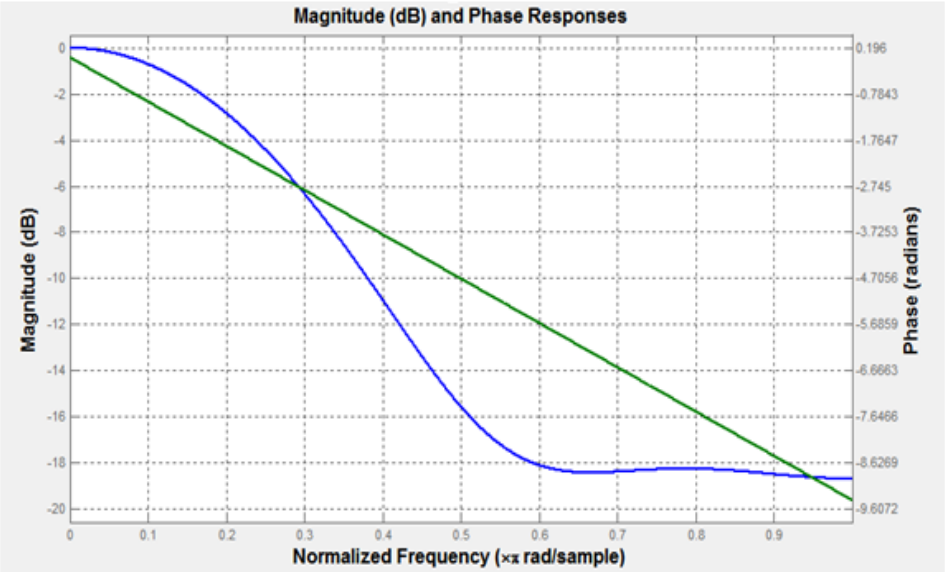


Fig. 18. Module response for the equivalent 8 order filtering. Note the linear phase design (final phase is 0).

III.2 Defining trajectory signature

A set of four primitives has been chosen to form the signature. By definition every primitive has the same length as the trajectory it describes.

We chose this approach because it has two potential benefits. Breaking the 2D data into four 1D vectors certainly increases the amount of information, but analyzing similitude in 1D sequences is easier than in 2D vectors. Also, the main goal of this project is comparing apparent motion between videos, so we do not want to lose geometric information of these trajectories. Using geometrical descriptors helps preserving such information.

The selected primitives are:

- Speed
- Acceleration
- Curvature
- Curvature derivative

Here we briefly describe how we computed these primitives.

Let a trajectory be defined by$\Gamma(t) = \{x(t), y(t)\}$. The dot notation stands for derivative, double dot for second order derivative.

III.2.1 Speed

Speed, by definition, is calculated as the distance variation between two samples:

$$s(t) = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} \qquad \text{Eq. 9}$$

III.2.2 Acceleration

Acceleration is calculated as the first order derivative of speed:

$$a(t) = \dot{s}(t) \quad \text{Eq. 10}$$

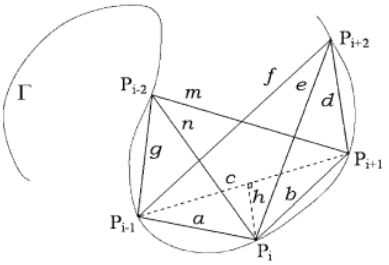
III.2.3 Curvature

To compute the curvature we used its differential definition (Eq. 11). As it relies on second order derivatives it is highly affected by noise, the adopted solution to that problem is filtering signals before each derivative computation.

$$k(t) = \frac{|\dot{x}(t)\ddot{y}(t)+\dot{y}(t)\ddot{x}(t)|}{(\dot{x}(t)^2+\dot{y}(t)^2)^{3/2}} \quad \text{Eq. 11}$$

There are however other interesting approaches to compute curvature, two were tested and they initially proved better than the differential approach because of the lack of previous filtering, but the differential one seemed to render more consistent results with the appropriate filtering applied.

First contender was a purely geometrical, fast computing approach used in [32]. This one uses three successive points to calculate the curvature. It also benefits from not having to compute the second order derivatives.



$$k(t) = 4\frac{\sqrt{s(s-a)(s-b)(s-c)}}{abc} \text{ where } s = \frac{1}{2}(a+b+c) \quad \text{Eq. 1}$$

The second contender is an intuitive method based on the definition of curvature in a circle. Three or five points are used to solve a circle fitting, the inverse of the radius is then taken as the curvature for the middle point.

III.2.4 Curvature variation

Curvature variation is simply the derivative of the previously calculated curvature:

$$dk(t) = \dot{k}(t) \quad \text{Eq. 12}$$

Two example trajectories and their respective signatures are provided in Fig. 19 and Fig. 20 respectively.

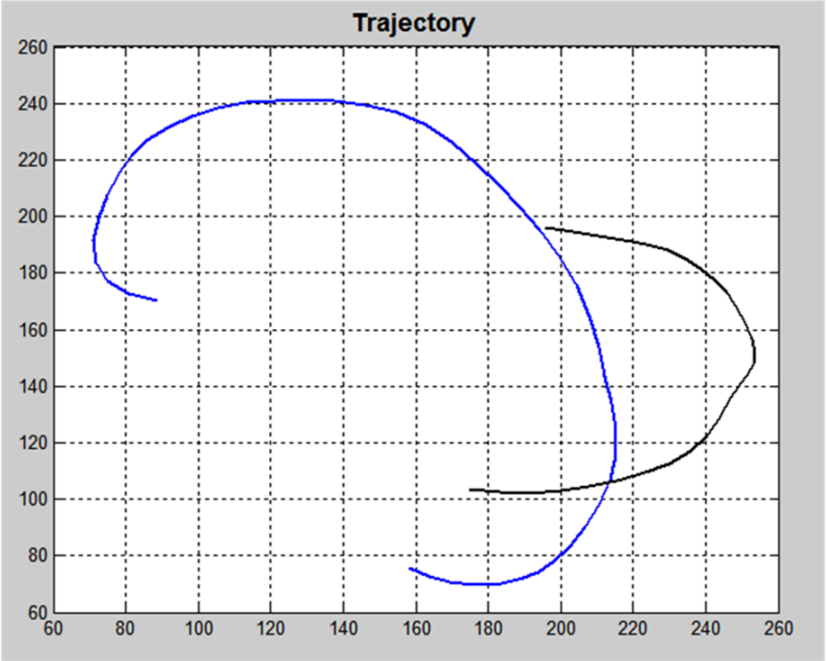


Fig. 19. Example of two trajectories with similar shape.

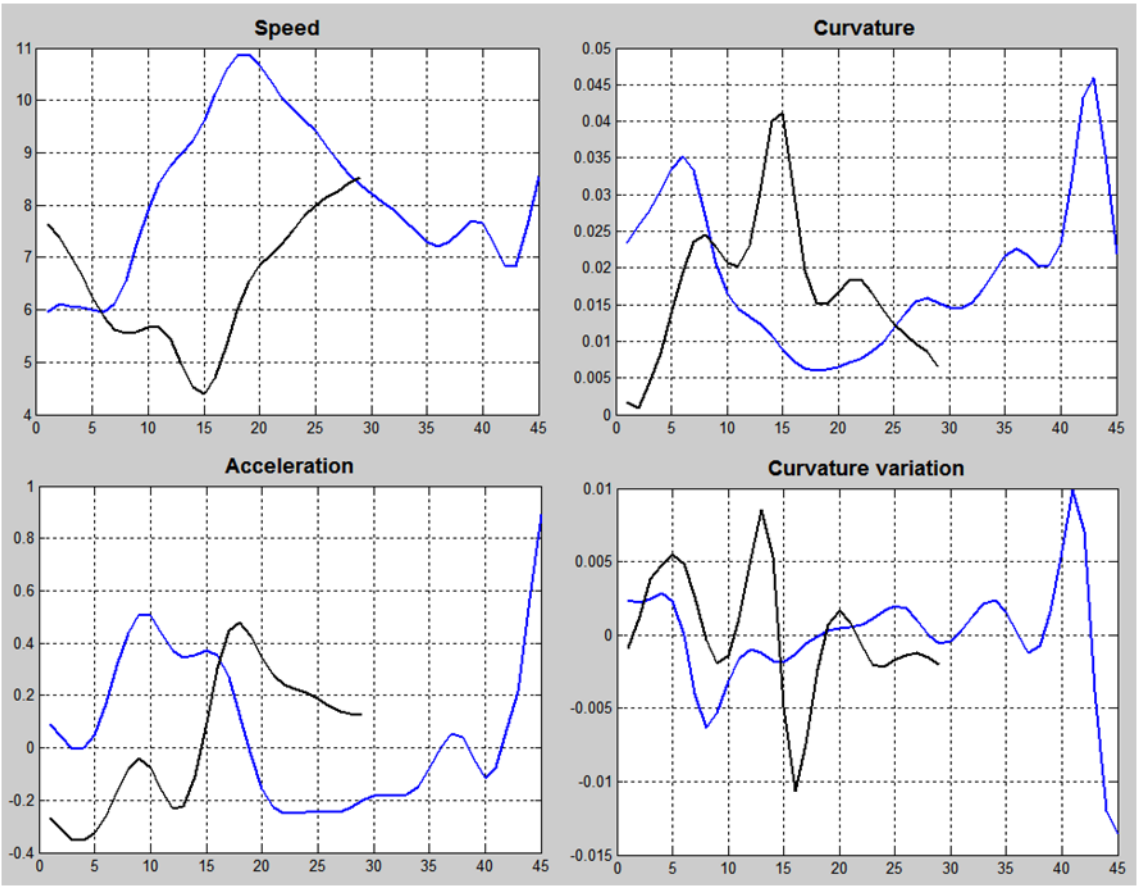


Fig. 20. Signatures of previous trajectory samples

III.2.5 Derivatives implementation

Noise has been a serious problem at extracting trajectories signatures because of the impact it has on derivatives. Because of this, several tests have been carried to determine how to proceed at calculating these derivatives.

- **Classic approach:**

  This estimate is computed as the variation between previous and next samples, it is a widely used formula. This equation is not valid for the first and last samples of the sequence, for these the direct difference of two consecutive samples is used.

$$\dot{x}_n = \frac{x_{n+1} - x_{n-1}}{2} \qquad \text{Eq. 2}$$

- **Modified 3 points:**

  This variation takes the average of the slope of the line through the point in question and its left neighbor, and the slope of the line through the left neighbor and the right neighbor. It is supposed to be more robust to outliers than any estimate considering only two points. Like in the classic approach, first and last samples are taken as the direct difference of two samples.

$$\dot{x}_n = \frac{1}{2}\left(x_n - x_{n-1} + \frac{x_{n+1} - x_{n-1}}{2}\right) \qquad \text{Eq. 3}$$

- **Interpolated 2 points consecutive difference:**

  This implementation is a personal test based on the intuition behind the derivative concept. The idea was solving the problem of the basic matlab derivative function (diff), which reduces the length of the output by one and shifts the result by a half sample. Direct difference between consecutive samples is calculated, this gives an approximation of the derivative between the two points. Cubic interpolation is then used to calculate the derivative at the location of the original samples. This process is illustrated at **¡Error! No se encuentra el origen de la referencia.**.

$$diff\_x(k) = x(k+1) - x(k)$$
$$\dot{x}_n = interp(diff\_x)_n \qquad \text{Eq. 4}$$

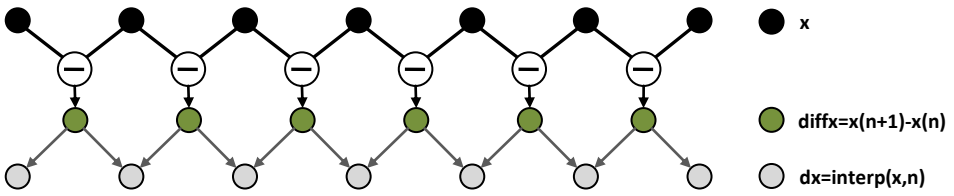


Fig. 21. Derivative by consecutive difference interpolation

These methods have been tested with both synthetic and real data from videos trajectories. Differences are not notorious, so we stick to the classic approach as it is the least computationally costly.

Anyway, as can be seen in the examples in Fig. 22, the key is using data which has been adequately pre-filtered because high frequency increases derivative variance dramatically for all the methods tested. For second order derivatives, the classic method is carried twice, but filtering after the first pass is applied.

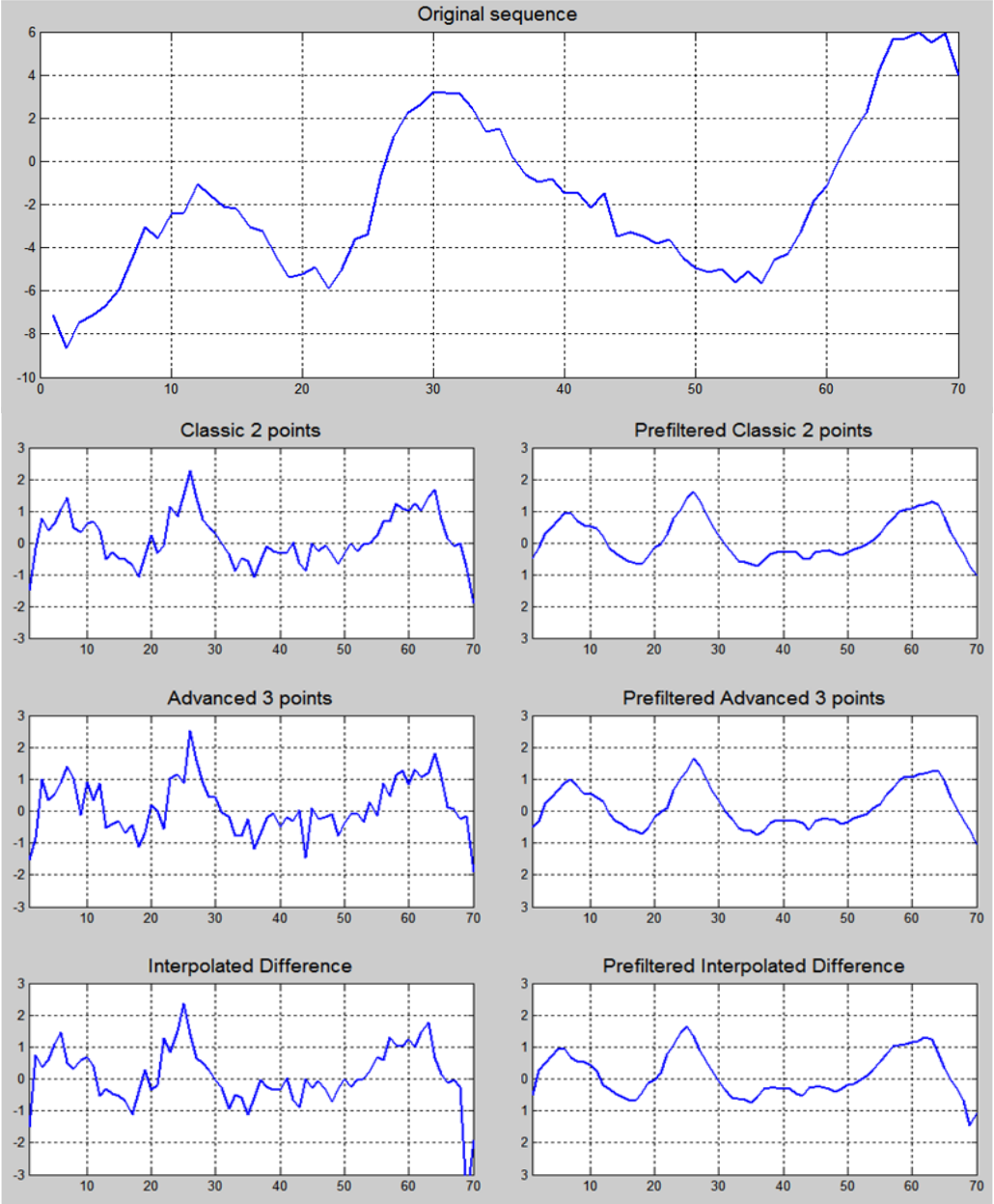


Fig. 22. Discrete derivative test. Derived signal on top. Left column shows direct derivative using the explained methods, right column shows the same methods but with prior signal low-pass filtering.

III.3 Trajectory component splitting

Under some circumstances trajectories may show erratic behavior at a given points. The path tracking algorithm explained in II.7 is prone to link two different motions under some circumstances (coming close or occlusion), thus linking paths of different nature into a single trajectory. Also, sometimes an object may vary its behavior at a given time, showing two separate motions. Finally, an error in the camera panning estimation would cause a generalized position jump in the trajectories, which would propagate into their signatures, especially into the primitives that are computed via higher order derivatives.

In a first approach, trajectories were cut based on searching for abnormally high curvature points. This however may cut trajectories which simply have almost linear motion and decreasing acceleration: for example a vertically bouncing ball has an infinite curvature point when it reaches its highest point. Using acceleration to detect cutting points has been tried as well, but this leaves outliers at other primitives. A combined decision criteria has been tested, but the parameterization proved very difficult because the gap between over-splitting and under-splitting was usually small and very dependent on each video trajectory.

The solution for this problem, which has proved to give far better final results, is cutting primitives separately instead of cutting trajectories. To detect cutting points a threshold has to be set for each primitive. Using a fixed set of thresholds is problematic because of the different nature of videos and because different previous normalizations may be done to the extracted paths. In the end this can be seen as an outlier detection problem.

Two set of thresholds are used for each trajectory. One is based on the information of the trajectory primitives alone and another based on the general behavior of all trajectory primitives in the video. The general set has sense under the premise that motion in each video is relatively coherent. The combination of both sets has shown to contribute to the robustness of the results. This is especially true when trajectories are short, because in this case the reduced number of data samples may lead to inaccurate estimations of the appropriate thresholds. This happens usually on short “noise paths” where the specific primitive calculated thresholds have too high values, in this case the general threshold set gives an upper bound.

Two methods for computing those thresholds have been tested. The first one relies on the assumption that component data PDF models are know (inferred by empirical observation), parameter models are estimated and the threshold is fixed using the Cumulative Distribution Function. The second one uses a simpler percentile based thresholding. Despite being the first method more fancy,

the second one proved to be more robust, hence in the final implementation we used the method described in III.3.2.

III.3.1 CDF estimation based thresholding

Empirical observation shows that for most trajectories, speed and acceleration values follow Gaussian like distributions, while curvature and its derivative fit better log-normal distributions. Note that acceleration and curvature derivative are not restricted to positive values; hence computing of the model is done on their absolute values. Some example of signature PDFs are shown in Fig. 23.

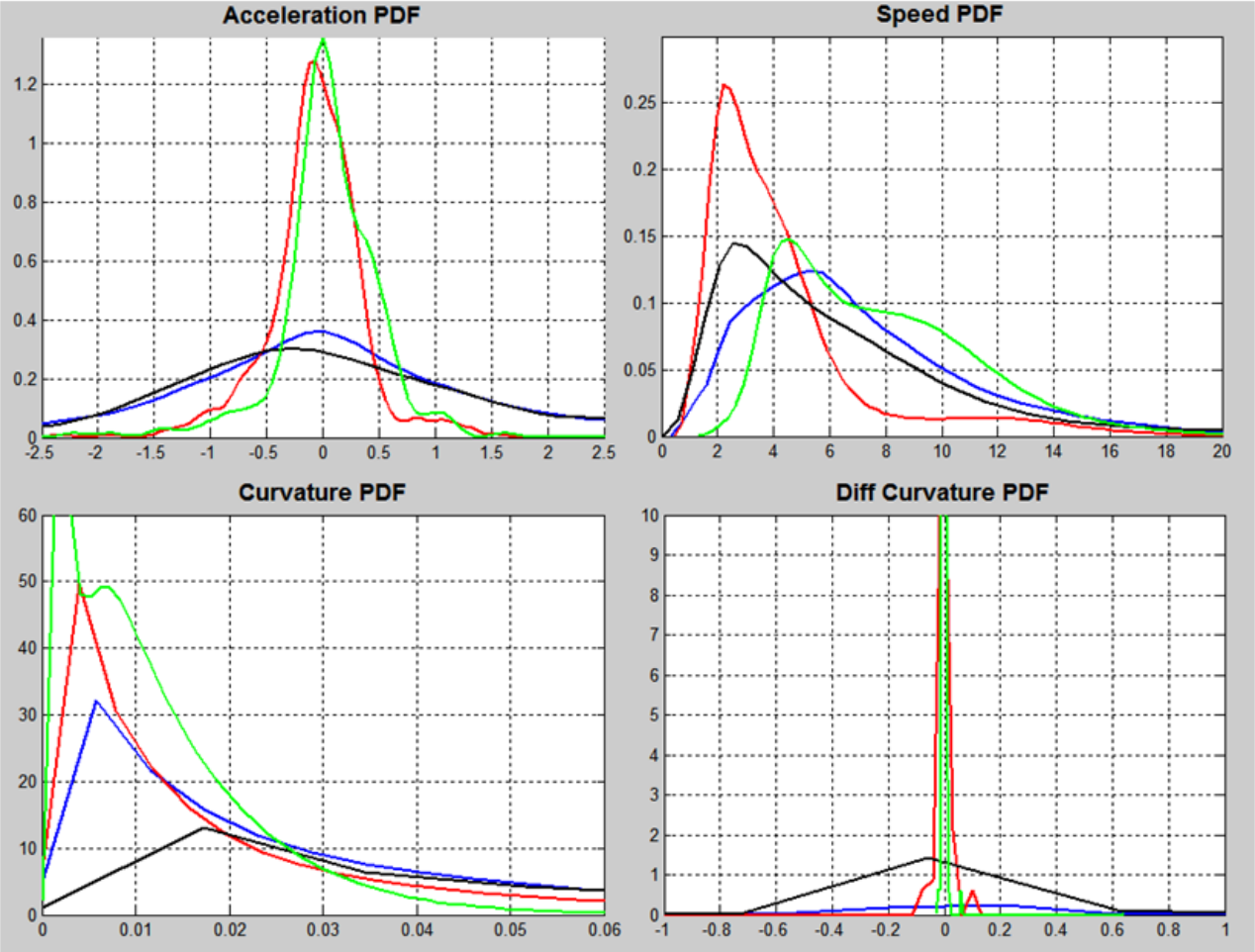


Fig. 23. Some video signature PDFs. (blue: soccer, red: highway surveillance, black: foreman shot, green: synthetic)

Threshold is calculated as the value at which the CDF exceeds a parameter α. Eq. 13 and Eq. 15 are the CDFs for Gaussian and Log-Normal distributions with parameters μ and σ respectively, Fig. 24 show examples of these distributions. Thresholds are calculated as in Eq. 14 and Eq. 16.

Parameter α is typically set at 95% for individual trajectory threshold and to 99% for general threshold. The α parameter for the general threshold is higher because it is used as an upper bound, if the same was taken, then many trajectories would be over clipped. Individual thresholds are calculated

using data from the primitive to be analyzed while general thresholds are calculated using all the signatures primitives of the given video.

$$F_k(k;\mu,\sigma) = \frac{1}{2}\left(1 + erf\left[-\frac{k-\mu}{\sigma\sqrt{2}}\right]\right) \quad \text{Eq. 13}$$

$$th_N = \mu + \sigma\sqrt{2}erf^{-1}(2\alpha - 1) \quad \text{Eq. 14}$$

$$F_k(k;\mu,\sigma) = \frac{1}{2}\left(1 + erf\left[-\frac{\ln k-\mu}{\sigma\sqrt{2}}\right]\right) \quad \text{Eq. 15}$$

$$th_{LN} = e^{\mu+\sigma\sqrt{2}erf^{-1}(2\alpha-1)} \quad \text{Eq. 16}$$

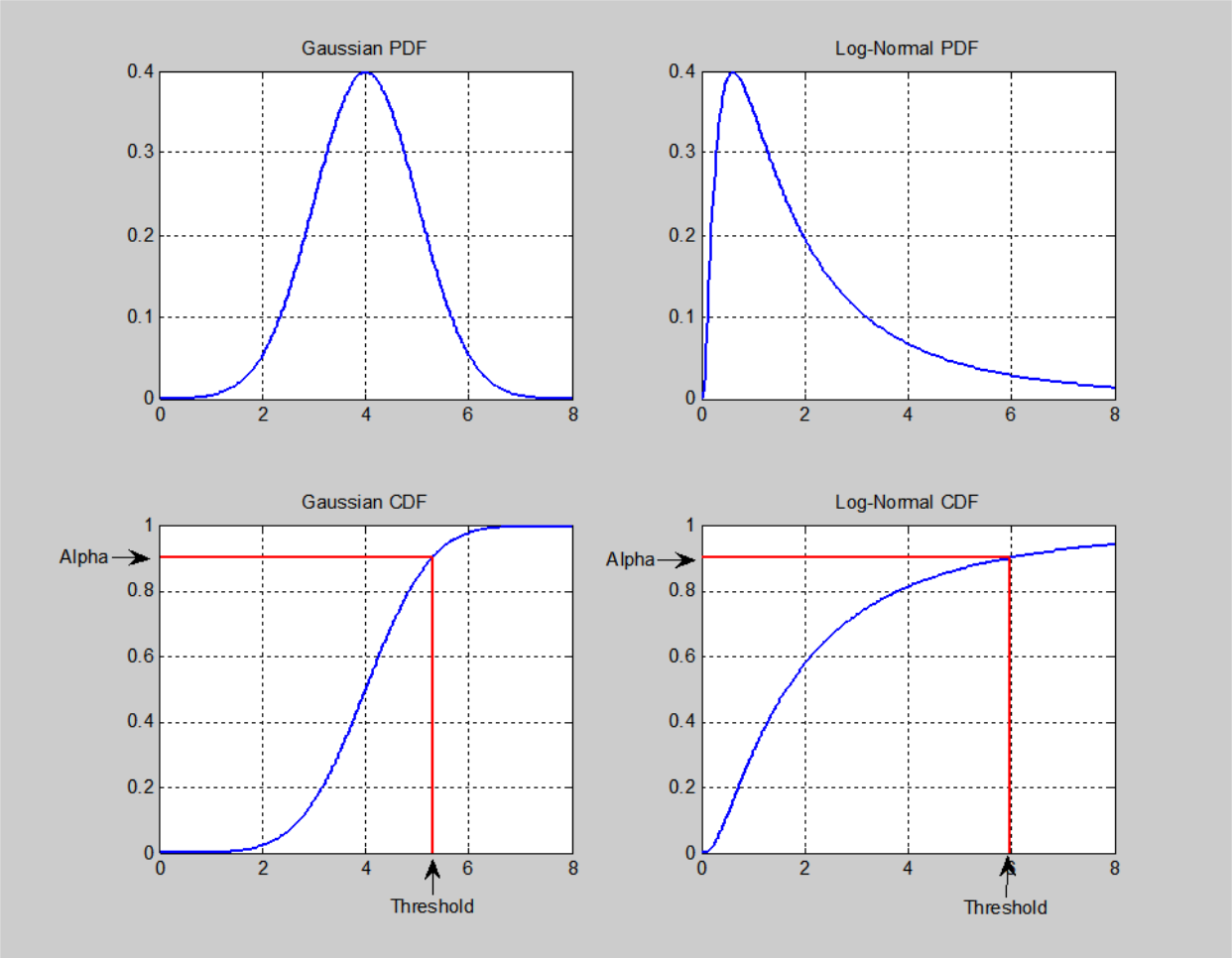


Fig. 24. PDF and CDF examples of Gaussian and Log-Normal distributions. Threshold calculations are illustrated, note α is set here at 0.9.

III.3.2 Percentile based thresholding

In this approach, which is the selected one for the final implementation, thresholds are computed by calculating a given percentile and multiplying it by a factor k. In practice, as data is discrete and generally its length is inferior to 100 samples, the percentile is calculated as follows:

- Component data is sorted
- Indexes for closer corresponding percentile are calculated as in Eq. 18 and Eq. 19
- Percentile is calculated as a weighted average of the upper and lower samples, being the weights the distance to the percentile, as in Eq. 20.

$$th = P_i \cdot k \quad \text{Eq. 17}$$

$$ind_{up} = ceil\left(\frac{n \cdot i}{100}\right) \quad \text{Eq. 18}$$

$$ind_{low} = floor\left(\frac{n \cdot i}{100}\right) \quad \text{Eq. 19}$$

$$P_i = \left|\frac{n \cdot i}{100} - ind_{low}\right| X(ind_{low}) + \left|\frac{n \cdot i}{100} - ind_{up}\right| X(ind_{up}) \quad \text{Eq. 20}$$

Percentiles and multiplying factors have been tuned by testing. Values chosen are detailed in the table below.

	Speed	Acceleration	Curvature	Diff. Curvature
Individual percentile	70%	70%	80%	80%
Individual factor	3	3	4	4
General percentile	80%	80%	85%	85%
General factor	3.5	3.5	4.5	4.5

The CDF estimation method is certainly more elaborate; however it does not work as well as the simpler percentile alternative. There are two main reasons:

1. CDF estimation is based on the assumption that distributions are known, which is not always correct; some videos differ too much from the models used here. While the percentile approach is more insensitive to the data distribution.
2. Outliers are taken into account when estimating the parameters of the CDF. As some of them have very high values they modify the estimated parameters of the distribution to some extent. This is especially true for the curvature component and its derivative, as a few

big outliers dramatically increase the computed mean and variance of the Log-Normal distribution.

Because of this, selecting an adequate α cutoff parameter is not possible: the correct value is too dependent on the number of outliers and their values and behavior would not be consistent. The percentile approach instead just does not take the outliers into account at all, provided the chosen percentile has not reached these outliers, but this is easily controlled by picking not so high percentiles.

III.4 Spatio-temporal and scale invariance

One key point of this work is comparing video motion structure in a general way, independently of exact location, rotation, scale and frame rate. It is clear this cannot be accomplished with raw data as even a slightly modified video would not match with itself at all. To deal with this problem two main approaches have been tested. The first relies on normalizing trajectories even before computing the signatures. The second leaves trajectories untouched, but normalizes signatures' primitives individually. Depending on the video source the final performance of different approaches vary, but with our final set of natural videos normalizing at component level to zero mean and unit variance gave the best results, the comparative results are presented in section V.3.

III.4.1 Raw trajectory normalization

Conducting geometrical trajectory normalization was used by **F. Bashir et al.** in [17] for trajectory indexing and retrieval. This normalization gives spatial position and scale invariance by shifting paths so they start at the origin and making them fit in a "unit box" (Eq. 21).

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \; ; \; Y' = \frac{Y - Y_{min}}{Y_{max} - Y_{min}} \qquad \text{Eq. 21}$$

This normalization effectively fits all the trajectories in a unit cube, but it does not scale both axes by the same factor, that clearly affects the primitives. For example an ellipse would transform into a perfect circle, which has constant curvature while the ellipse does not. Our proposal is avoiding this inconvenient by modifying denominators to equal the minimum scaling of any of the two dimensions (Eq. 22), this way proportions are kept constant.

$$X' = \frac{X - X_{min}}{\min(X_{max} - X_{min}, Y_{max} - Y_{min})} \; ; \; Y' = \frac{Y - Y_{min}}{\min(X_{max} - X_{min}, Y_{max} - Y_{min})} \qquad \text{Eq. 22}$$

It is a matter of discussion to which point such normalization should be applied systematically for general purpose videos. Let two trajectories s_1 and s_2 belonging to the same class of motion

predominantly a rectilinear motion, but S_1 lasting much longer in time than S_2, S_2 will then be spatially shorter. S_1 will be downscaled more than S_2, hence their respective signatures will still have similar shape behavior, but different average values (scaling would extend to the signature).

III.4.2 Primitive level normalization

This approach involves leaving the trajectories untouched and then normalizing the signatures primitives independently. Several normalizations have been tested and some were discarded, finally the ones that proved to be useful are:

- **Power**: scales the sequence to make its average power equal to one (Eq. 23) .
- **Mean and variance**: offsets and scales the sequence to force zero mean and unit variance (Eq. 24).

$$y_i = \frac{x_i \cdot \bar{\bar{x}}}{\sqrt{\sum x_i^2}} \qquad \text{Eq. 23}$$

$$y_i = \frac{x_i - \text{mean}(x)}{\text{std}(x)} \qquad \text{Eq. 24}$$

Zero mean/unit variance was the average winner, but by a narrow margin, also in a number of video sets the power normalization gave better results, so it cannot be concluded which one is more adequate, more tests should be carried. Specific results are provided in section V.3.

III.5 Inter component distance metric:

The most elemental way of comparing two sequences X and Y is using Euclidean distance along all its elements:

$$d = \sum_i \sqrt{{x_i}^2 + {y_i}^2} \qquad \text{Eq. 25}$$

However, that norm is not useful to build a metric for sequence similarity because it does not take into account general shape, in fact the distance between a sequence and the same sequence shifted slightly may be high. Also, such a norm can be applied only to equal length sequences.

To compare time series many techniques have been used, here we chose to base our algorithm on the philosophy of Dynamic Time Warping (DTW). The goal of this algorithm is comparing two sequences that have different local speeds and time shifts by aligning them dynamically in the time axis. Here, a heavily modified DTW algorithm has been iteratively used to find the best match between two sequences (trajectory signature primitives).

The metric we propose has been called “Multi-Scale Sliding Window CDTW” (abbreviated MDTW). To expound how it work, we first explain the basic DTW and a variant known as DDTW, then a novel implementation we called CDTW is proposed, this one is based on the same principles but combines the strong points of both DTW and DDTW . Finally the MDTW algorithm that uses our CDTW to map a component into a sub-sequence of the compared component is detailed. This way we can find the best match of two sequences in terms of local scaling and offset to determine an accurate distance in terms of shape similitude.

The motivation of using this approach is giving some degree of flexibility at comparing signature components that more basic metrics do not allow (like Lp norms). But at the same time we want to assign low distances only to signatures that have similar shape, which may not happen when using other techniques. As counterexample, let’s suppose we measure distances in the DFT domain. If we pick a signal, transform it to the frequency domain, and modify slightly the phase while maintaining the modulus intact, we will obtain a new signal which distance under such metric may still be low compared to the original, but once we return it to the time domain its shape may have varied substantially. To illustrate this effect we show the change in shape of a sequence which phase has been modified linearly in Fig. 25. Phase in frequency domain as been multiplied by a complex exponential$e^{j\alpha f}$, no 2π phase jumps are present.

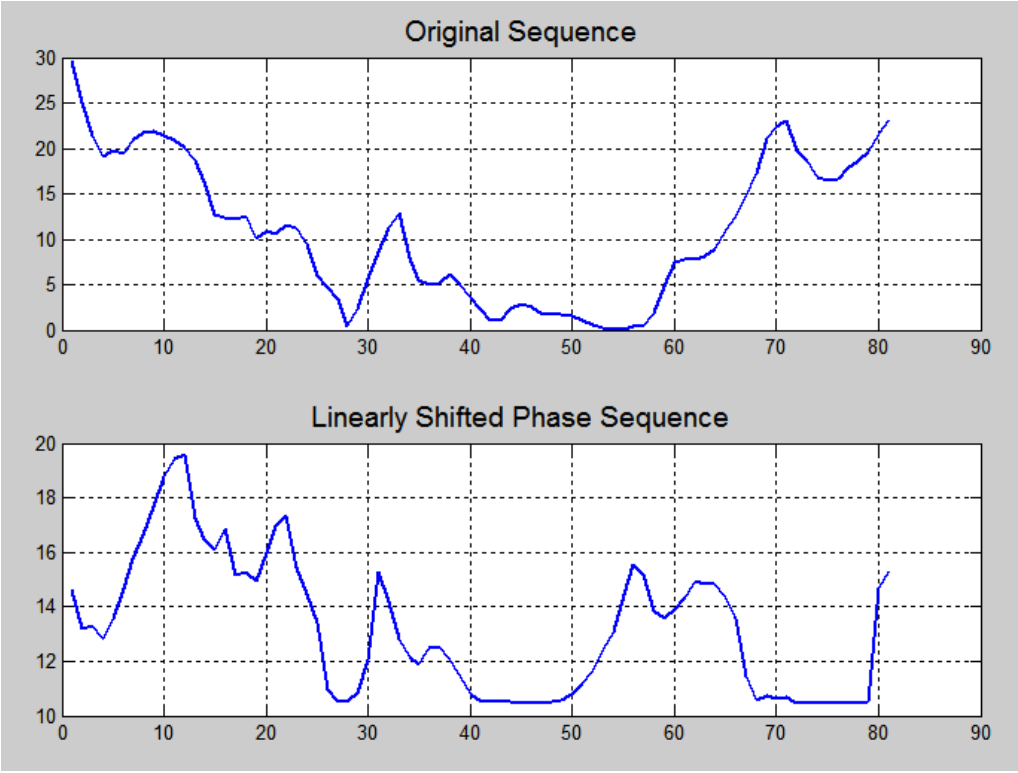


Fig. 25. Plots of a sequence before and after having its phase linearly modified.

III.5.1 Dynamic Time Warping algorithm (DTW)

DTW [33] is an algorithm used for measuring similarity between two time-series that may vary in time or speed [34]. It has been employed in speech analysis [35] and later extended to other purposes, including trajectory indexing and retrieval [17]. It is especially useful because of its simplicity and its ability to cope with missing data and non linear variations in the time axis.

DTW still relies on a classic distance operator, typically an Lp-norm, which is used to search for optimal alignment between the two input time series via minimization of a cumulative distance across samples. The distance between two series, X of length N and Y of length M can be measured by constructing a warping path W:

$$W = w_0, w_1, \cdots, w_k \qquad max\{N, M\} < k < (N + M - 1) \qquad \text{Eq. 26}$$

K is the length of the warp path $W_K = (i,j)_K$, where (i,j) are the matching indices for X and Y (temporal alignment).

An NxM matrix is built where each element contains the distance between $X_i$and Y_j elements, this matrix is used to search the lower distance cost warping path.

The warping path is subject to a set of constraints:

- Start and finish conditions: $W_1 = (1,1)$ and $W_K = (M,N)$ as the warping path must cover both input sequences.
- Continuity: W does not jump in time index.
- Monotonicity: W has to be monotonically spaced in time.

There is a huge number of possible warping paths, however we are only interested in those that minimize the warping cost:

$$DTW(X,Y) = min\left\{\frac{\sum_{k=1}^{K} d(w_k)}{\mathrm{K}}\right\} \qquad \text{Eq. 27}$$

Where d is the distance between ith and jth elements of input sequences for the kth warp and K denominator is used to normalize for different warp path lengths.

Fortunately thanks to the constraints a full search is unnecessary, cumulative distance $\gamma(i,j)$ is iteratively found as the distance $d(i,j)$in the current cell and the minimum of the cumulative distances of the adjacent elements (allowed by the constraints):

$$\gamma(i,j) = d\big(X_i, Y_j\big) + min\{\gamma(i-1,j-1), \gamma(i-1,j-1), \gamma(i-1,j-1)\} \qquad \text{Eq. 28}$$

This basic DWT algorithm has limitations when it comes to matching sequences that have similar shape but local variations in magnitude, thus producing singularity points in the warping (a single point on one sequence maps onto a large subsection of another sequence), this phenomenon can be seen in Fig. 27. Additional constraints can be added to try to avoid this problem, these includes using a warping window that would not let the path separate much from the diagonal as used in [34] and [36], fixing a step constraint which avoids too many points from a sequence from being assigned to just one point of the other [37] or using slope weighting to bias the path towards the diagonal [38].

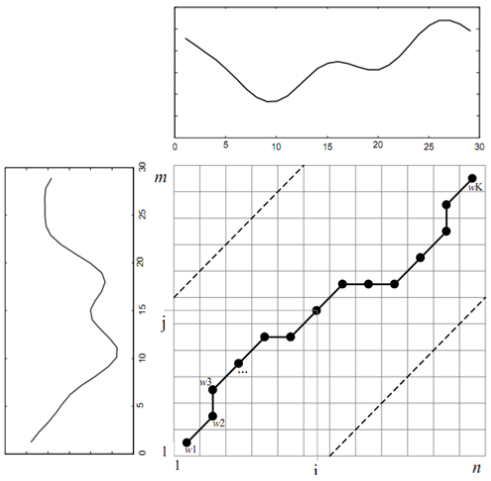


Fig. 26. Warping path example

III.5.2 Derivative Dynamic Time Warping (DDTW)

Keogh and Pazanni [39] introduced a modification to the DTW which reduces significantly the singularity problem DTW suffers and hence returns a more "accurate" warping.

The main weakness of DTW is it only considers raw values of the sequences, so it will treat equally two points with identical values but different local trends (one rising, the other falling). To prevent this the DDTW uses a higher level feature that takes into account "shape", so instead of using direct distance between sequence elements, the metric is applied to the derivative of the sequences. Two examples are shown in Fig. 27, here signals have the same phase but differ locally in height. Here, while DTW generates spurious warping at problematic points DDTW finds a solution much closer to an intuitive warp.

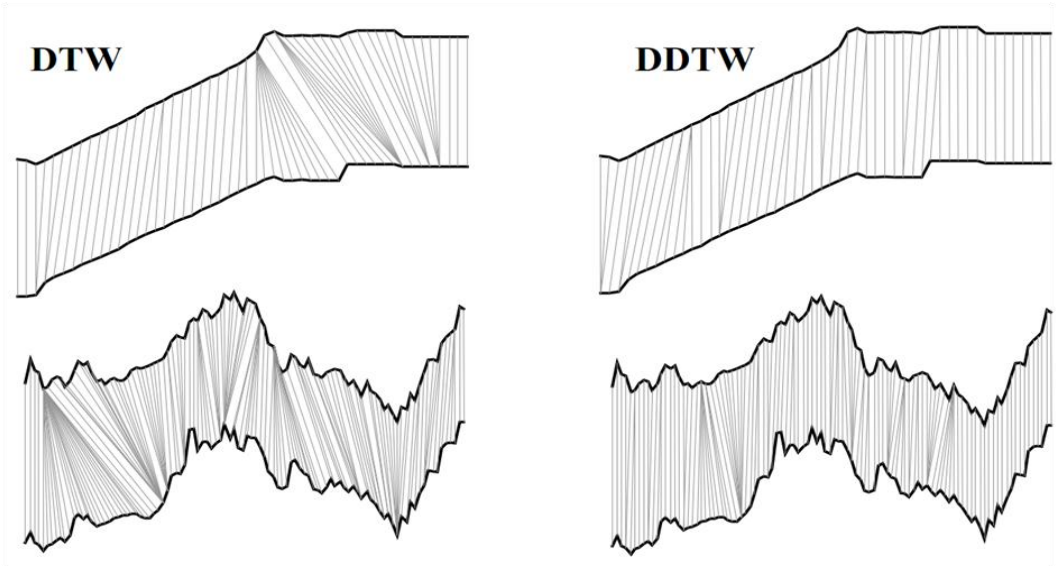


Fig. 27. DTW and DDTW example comparison from [39]. Sequences are not warped, but minor differences in height causes wrong "spurious" warping assignments with DTW algorithm

III.5.3 Proposed metric: Combined Dynamic Time Warping (CDTW)

DDTW generally outperforms DTW, but using only the derivative to build the distance matrix is not optimal when facing truly different input signals as it is the case in this work. In our approach both raw and derivative distances are combined.

In first tests, the direct distance and the derivative distance matrices were directly combined using weights. Results are not bad, however after inspection it was seen that depending on the type of input the signal power relationship between direct distance and the derivative distance may vary considerably: by simply increasing the mean difference between input sequences the direct distance increases as well, thus giving higher bias towards the direct distance matrix when choosing the warping path. Because of this it is not possible to correctly weight the influence of direct and derivative distances for general use.

The solution we adopted is normalizing the power of both distance matrices prior to combining them. This way the weights applied really give control over which distance matrix is predominant in the warping path decision, and more important, it stays the same regardless of the input sequences.

Normalization is done as in Eq. 29, where D is a MxN distance matrix and D_N is its power normalized version. The final D_{WP} matrix used for searching optimal warp path is computed as a weighted sum of the two normalized distance matrices $D_{N\text{-}Direct}$ and $D_{N\text{-}Diff}$ as in Eq. 30. Values chosen as weights in this work are $\alpha=1$ and $\beta=0.5$.

In this implementation the distance matrix D_{WP} is only used to search the warp path, the final distance between the two input sequences is however calculated using the direct distance matrix alone. A block diagram of the algorithm is shown in Fig. 29.

$$D_N(i,j) = \frac{D(i,j)}{\sqrt{\frac{\sum_{i=1}^{M}\sum_{j=1}^{N} D(i,j)^2}{M \cdot N}}} \qquad \text{Eq. 29}$$

$$D_{WP} = \alpha \cdot D_{N-Direct} + \beta \cdot D_{N-Diff} \qquad \text{Eq. 30}$$

From now onwards, the algorithm will be abbreviated as CDTW (Combined DTW), not to be mistaken with **Efrat et al.** Continuous DTW [40], which is an extension of the DTW spirit to the continuous domain.

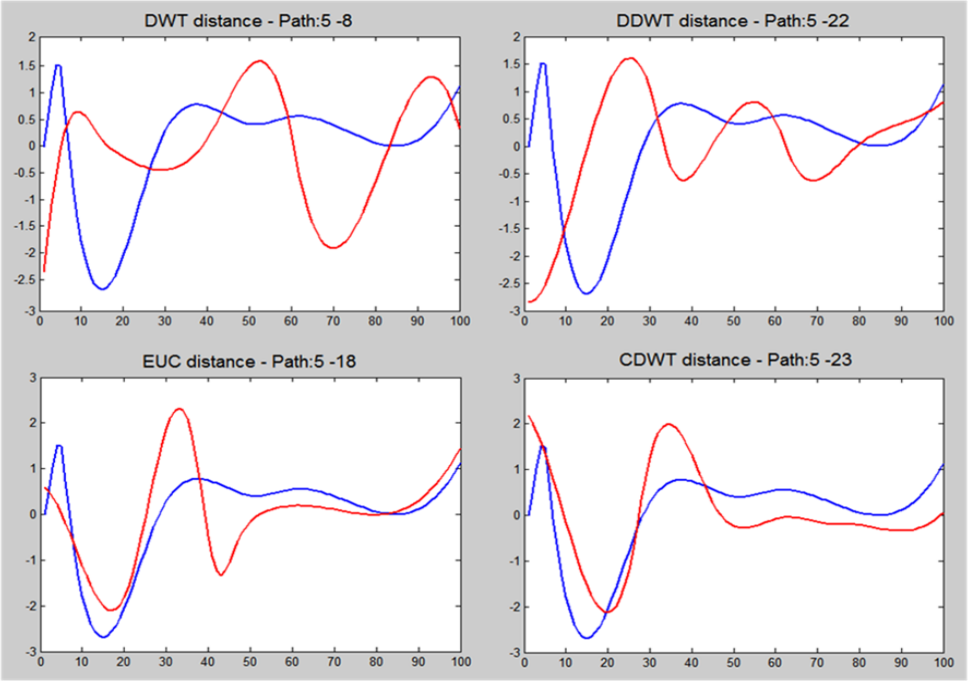


Fig. 28. Sequence matching test for different metrics. Selected method result shown at right bottom.

An example of sequence matching is show in Fig. 28. A set of 25 pseudo-random sequences have been created, then distance between each other sequence has been computed using Euclidean distance, DTW, DDTW and the CDTW metric presented here. One sequence is picked as query (plotted in blue), then for each metric the best match is presented (in red).

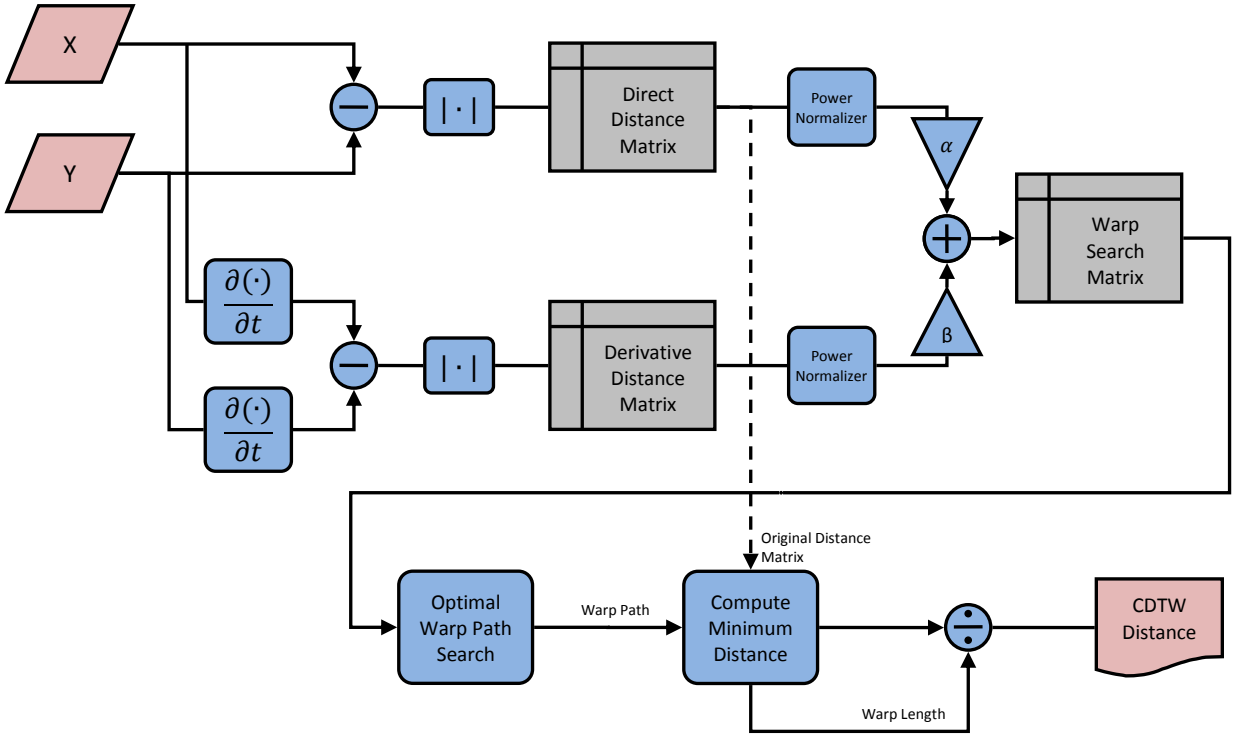


Fig. 29. CDTW algorithm

III.5.4 Sub-sequence mapping: Multi-Scale Sliding Window CDTW (MDTW)

Trajectories may have different lengths, CDTW can cope with a moderate size difference, but it cannot map a sub-sequence to a part or a sequence. CDTW tries to match the two input sequences in their full length because of the start and finish conditions. Breaking this constraint would solve partially the problem, but then deciding the starting and ending points would increase the problem dimensionality. Instead, a sliding window and multi-scale approach is proposed: the goal is iteratively finding the sub-sequence in the longest input sequence that better fits the shorter input one. In other words, a search for minimum possible distance between the shorter sequence and a sub-sequence of the longer one has to be found. A full search is potentially slow, so assuming this minimization has a local concave distance shape, an iterative search is proposed.

Let be X and Y two sequences to compare with lengths N and M respectively, with N<M.

- **Step 1:** X is compared using CDTW to windowed versions of Y, being W a uniform window of variable size around N. The window is shifted and scaled successively. Offset is modified by an initial step, typically 20% of the size of X. Scale is modified so the length of W goes from N/4 to 2N in 5 steps. The output D_{xy} is a matrix of size $(M-N)/0.2N \times 5$. The indices of the minimum of D_{xy} are used to determine the initial window offset and scale for best match between the two sequences.
- **Step 2:** The offset step is halved. Being the scale fixed, distances are calculated at plus and minus the offset step. The refinement distance vector has length 3, the lower distance is used to update best offset.
- **Step 3:** Like step 2 but for scale, maintaining offset fixed.
- **Step 4:** Steps 2 and 3 are iterated until a convergence criterion is reach. In this work we went down to the finest possible refinement. Note that convergence for offset and scale may have different speeds.

Besides from calculating the lowest possible distances between the two sequences, the algorithm also stores the minimum length of the two input sequences. This length will be used later for weighting purposes. We called this algorithm Multi-Scaled Sliding Window CDTW, to abbreviate we refer to it as MDTW. A diagram of the algorithm can be seen at Fig. 30

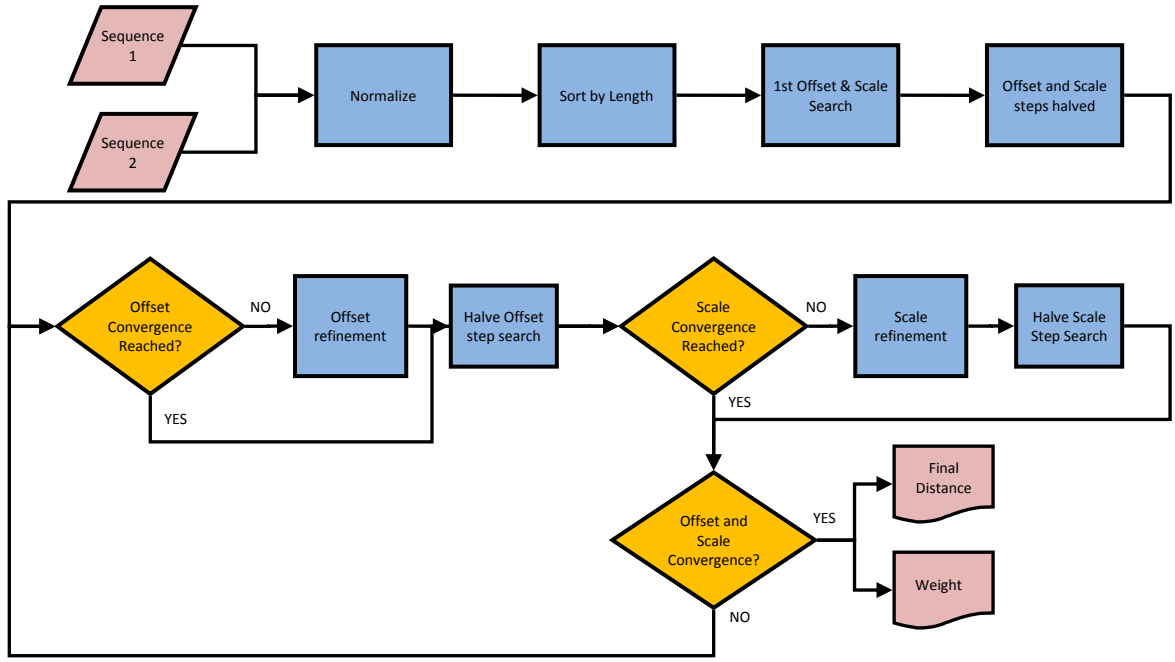


Fig. 30 Minimum sequences distance workflow

To illustrate how this algorithm behaves in a real case, an example is presented in Fig. 31. From a 270 samples sequence, 75 samples are extracted, then up-scaled to 115 samples and finally white noise is added. The resulting sequences are used to feed the algorithm. The distance matrix for the first offset and scale search is presented as a surface plot. To make the surface plot denser, the number of scale search points has been increased on purpose, in actual calculations this is reduced to save computing power. The original input sequences and successive iterations are shown along with the resulting distance evolution.

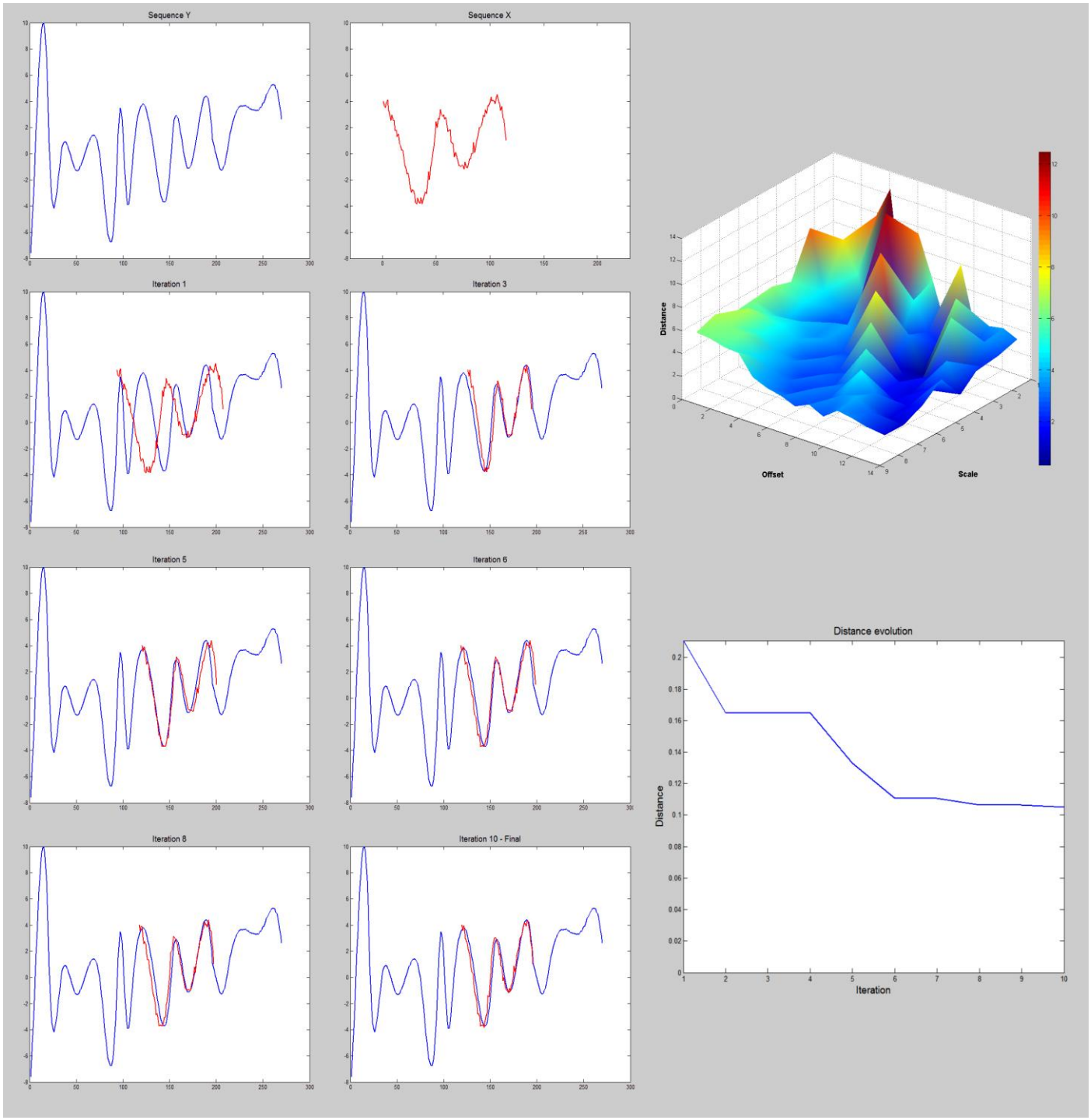


Fig. 31 Multi scale/offset distance calculating example. Left plots show input sequences along with several iteration adjustments. Upper-right surface plot shows the first step distance measurements. Down-right plot shows distances reduction against iteration number.

IV Video dissimilarity metric

In the previous section it has been explained how to split the video trajectories into a set of signature primitives and how these primitives were split into components at key points. Also, MDTW iterative algorithm used to measure the distance between two components has been introduced. In this section, three methods to calculate a final distance between two videos from a database are explained. The first one is a simplistic approach which mainly serves to validate some ideas; these ideas are developed later in a more sophisticated fashion using probability density estimations; finally a double pass algorithm is explained as the final adopted solution. All of them are presented here because they are successive evolutions with increasing complexity, so each method helps understand the next one, but the final implementation corresponds to the third one, called “Double Pass multi-weighted PDF”.

In IV.1 we detail how we compute distances between signatures components of two videos and their associated weight matrices that are used later to adjust the distances relevance. A basic method called “selective mean” to extract a final distance is explained in IV.2, this first approach illustrates the idea behind the subsequent PDF based implementation described in IV.3. The final algorithm based on the basic PDF implementation is expounded in IV.4.

IV.1 Building distance and length-based weight matrices

As described in the previous section, signatures are extracted from video trajectories and their primitives split into components. Every trajectory is thus converted into 4 sets of components, each set corresponds to the 4 kinds of primitives (speed, acceleration, curvature and curvature variation). Note for many trajectories its signatures primitives are not split, in this case a trajectory translates into exactly four components, but others trajectories may have its primitives cut and hence the number of components is greater than the number of originating primitives.

For each primitive type, the cross distances of the video components are calculated. That is, for every component in the first video, the distances to every component of the same primitive type in the second video are calculated.

The MDTW algorithm is used, which returns a distance alongside a weight based on the input components’ lengths for each comparison. The result for each video to video comparison is hence four distance matrices and four weight matrices. Those weights are stored in a separate set of matrices, that have exactly the same size as the distances ones. These matrices are the input data for the algorithms explained in this section.

Note that the data from different primitives is processed independently, so the algorithm is run four times, being fed with one distance matrix and its corresponding weights matrix at each time. Hence, four dissimilarity measures are obtained for each video to video comparison. How to combine this information is described later in this section. To illustrate the building of those matrices a diagram is shown at Fig. 32.

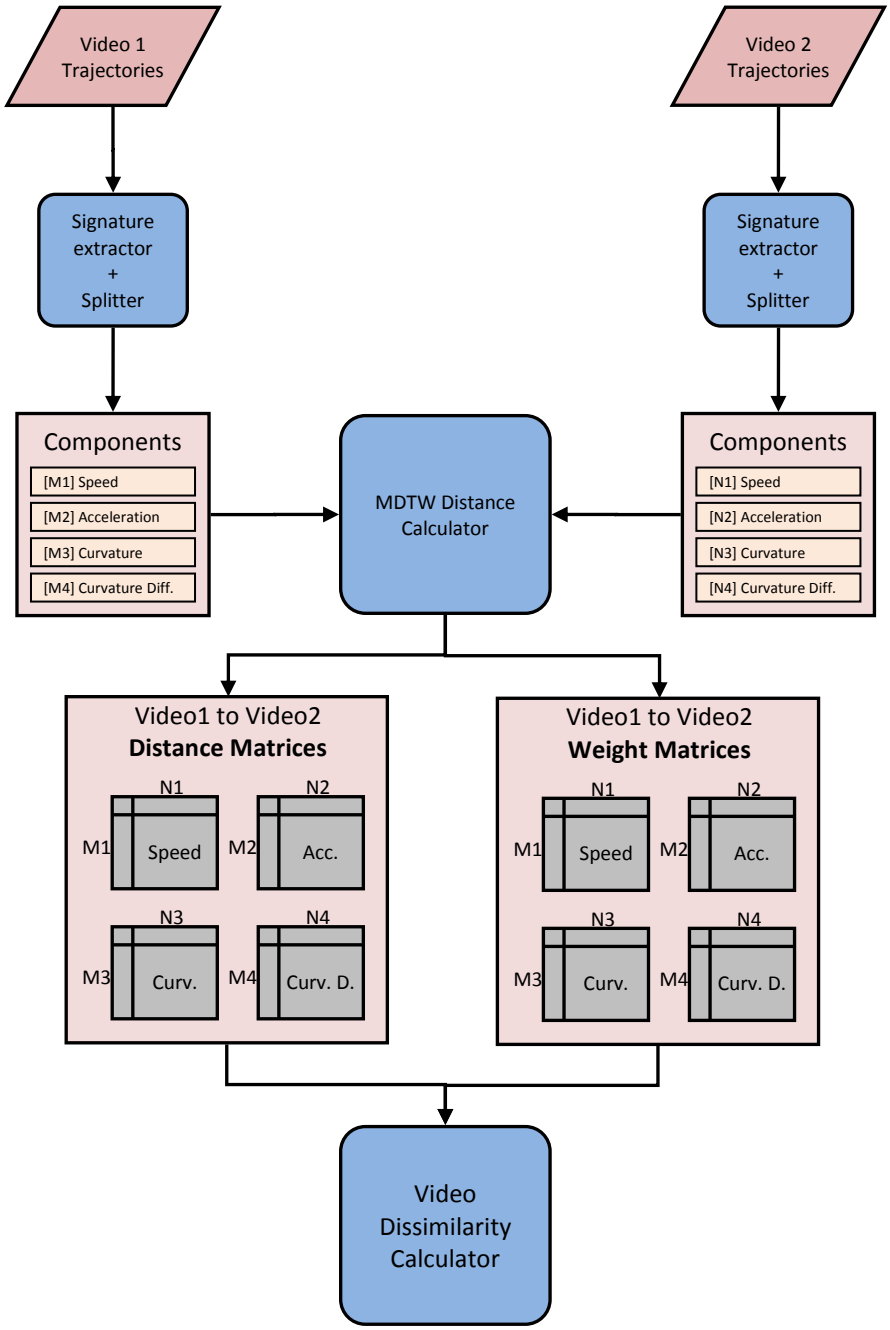


Fig. 32 Distance and Weights matrices building. The sizes of the matrices are indicated, these depend on the number of components for each video and primitive type.

IV.2 Algorithm 1: Selective mean

For testing purposes and prior to establishing this method, the direct mean of the elements of the distance matrices has been tried as a measure of dissimilarity. The problem with such metric is the distance between a video to itself could be high simply because the trajectories are different enough. The results are thus almost useless.

A more sophisticated idea is looking how each trajectory in one video match enough trajectories in the comparing video. A simple approach is considering only the best results (lower distances) for each set of comparisons, this way we only focus on the sets of trajectories that share similar motion to those of the comparing video. For every component, only a number of the lowest distances are taken into account. The process is simple: for every primitive distance matrix rows are ordered in ascending order, then the matrix is cut at a given column. The mean is then calculated with the remaining results. This method already gives decent results when tested on synthetic videos, it is useful because it illustrates the idea behind the final solution. However it did not prove to be accurate enough when dealing with real videos.

IV.3 Algorithm 2: Single Pass bi-weighted PDF peak search

With this algorithm the results from the component distance matrices are used to estimate probability density functions. Each component has a number of distance measures equal to the number of components present in the comparing video. The PDF estimated using these distances indicates the distribution in terms of distance to the components of the other video. How to exploit this is explained later.

The PDF estimations are carried using Matlab's ksdensity function included in its statistical toolbox. Ksdensity computes a probability density estimate of the input data. It is based on a normal kernel function with a window width that depends on the number of input data points, as described in [41]. This implementation has some useful features used in this work:

- **Support:** It restricts the density estimate range. As all the input data are distances (hence positive values) the support is restricted to $\mathbb{R}^+$. This gives a more accurate estimation close to the origin.
- **Weighting:** Allows assigning different weights to each input sample. This is a key feature as the algorithm heavily relies on weighting the input data.
- **Specific evaluation points:** Points where PDF has to be evaluated can be set externally. By default 100 equally spaced points are taken from the input data range. Here a different selection of evaluation points is used so we can obtain enhanced precision where needed.

IV.3.1 Component length and trajectory object size weighting

This algorithm uses length weights given by the previous MDTW block. Longer components results are more relevant than shorter ones, so higher weights have to be assigned to them and vice versa. As each distance measure involves two sequences, a decision has to be taken about which norm to use. The decision adopted has been taking the minimum of the two lengths, as the algorithms tries to match the shortest one into the longer one.

Aside from length weighting, it has been considered that there is a correlation between similar moving objects and the size of these. Tiny objects will probably share more motion with other tiny objects from a similar video and vice versa. Also, higher weights are assigned to objects which are bigger. To do such a function which takes into account raw objects size and size differences between the comparing components has been used.

The two sets of weights are combined by multiplying them before being applied. This weighting strategy based on two features (component length and object size) gives the “bi-weighted” prefix to the algorithm name.

IV.3.2 Using PDFs to establish a distance

Here, the mean of distances to the components of the comparing video are not used anymore. The idea is PDF maxima will be located where there is a high concentration of distance measures of relevant components from the comparing video. The first local maximum that exceeds a given threshold is selected and its abscissa is taken as the distance between the analyzed component and the components of the comparing video. This threshold is taken as a given percentage of the absolute maximum. The selection of such threshold is important, as it determines the selectivity of the algorithm: the lower the threshold, the least number of similar components in the comparing video will be needed to assign a low distance. We experimentally set this threshold at 50%.

This way if the two videos have enough components which are similar, the resulting distance will be highly biased for those matching components, reducing the effect of other secondary motions, which is the main purpose of this algorithm.

Using this method, for each component in the first video a distance to the components of the other video could be obtained, this way each distance matrix would collapse into a distance vector of length equal to the number of components of the first video. However, Instead of searching a distance for each component, a global distances PDF is obtained, this could be done by calculating individual components PDFs and combining them together, but this is unnecessary as we can directly calculate the global PDF using all the data from the distances and weight matrices. This has some advantages:

- Better global PDF fidelity: the higher number of input values the narrower the kernel smoother is, thus increasing the precision of the estimated PDF.

- The operation is commutative: the same data is feed into the estimator independently of the order of the inputs. Hence distance from video 1 to video 2 is the same than the distance from video 2 to video 1.
- Lower computational cost: each PDF estimation is done over a given number of points, computing fewer estimations reduces computational cost significantly.

Fig. 33 shows individual and global PDFs for two video comparisons. Only the speed signature PDFs are shown. Here a video containing objects moving with a sawtooth speed pattern is confronted to two other videos, one with sinusoidal speeds and the other with constant speed. As sawtooth and sinusoidal are more similar the distance between these videos is lower than when comparing the sawtooth and the constant speed videos. In the figure sawtooth vs sinusoidal speeds are shown in the upper left plot, while the sawtooth vs constant speed is presented in upper right plot. The axis scales has been fixed for easier comparison.

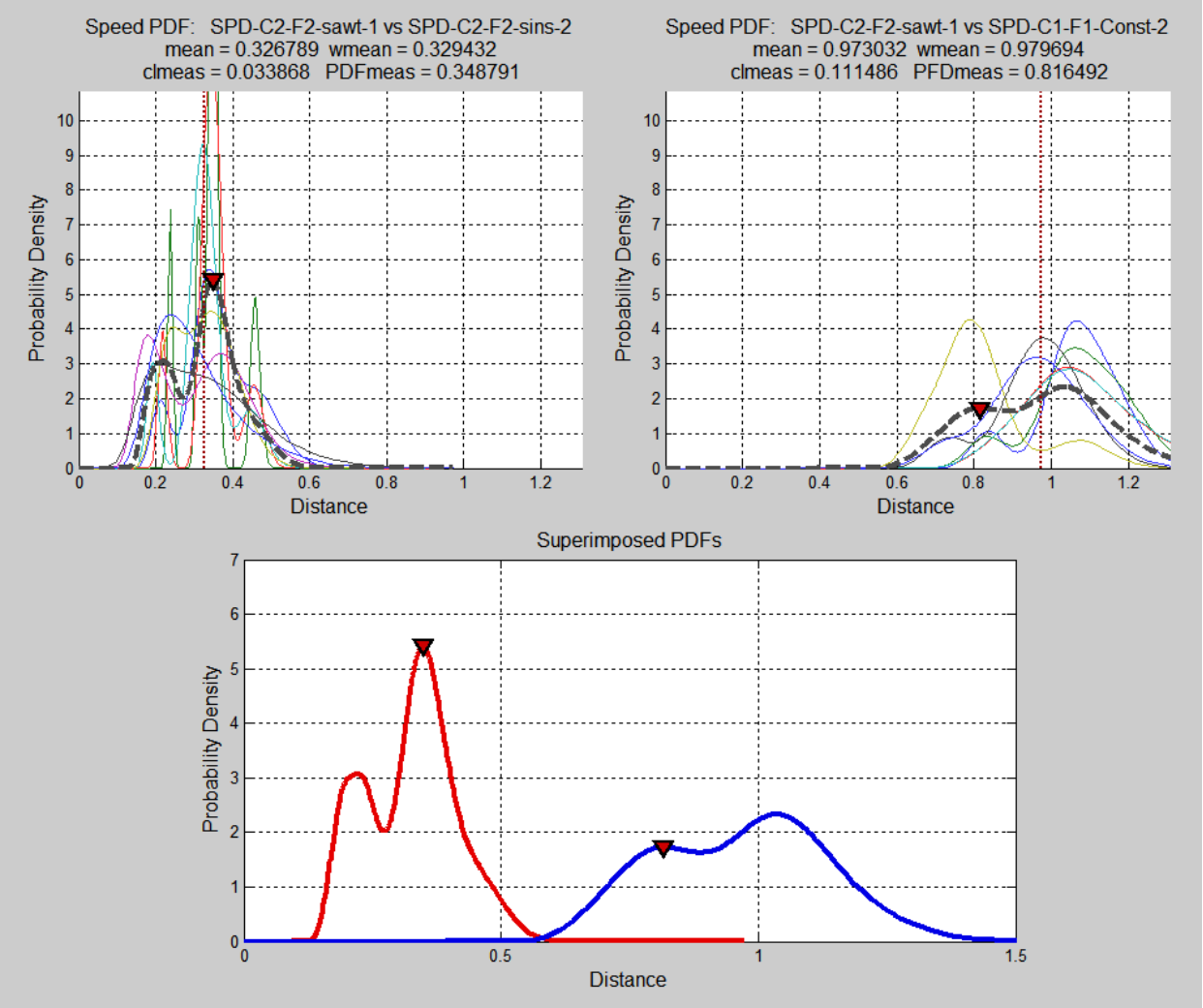


Fig. 33 Distance PDFs of one video against two others. In the first case (red), the speeds are more similar (sawtooth vs sinusoidal) and hence final distance is lower. In the second case (blue), the speeds behave differently (sawtooth vs constant) resulting in higher final distance.

IV.4 Algorithm 3: Double Pass Tri-Weighted PDF

The final solution relies on the previous one, being an evolution which takes into account an additional form of input weighting. Complexity is however increased, because the new weighting stage added is based on the results obtained by a first PDF analysis similar to the one proposed for the the previous method. Hence this algorithm uses a double pass strategy. An illustration of its workflow is provided in Fig. 36.

IV.4.1 First pass: Relevance weights computation

The purpose of this weighting stage is biasing measures from dissimilar videos towards higher values by re-weighting components that do not fit well into the compared video. The point is increasing the final distance between two videos when they do not share enough motion and viceversa.

To do such a first pass is carried in a similar fashion as in the previous method. Using the components' distance matrix, the PDFs of the inter-component distances are estimated and the distances are extracted. This time the computation has to be done on a per component basis and not using the whole data distance matrix because we need the individual distance for each component.

The means of these distances are calculated, with proper length/size weighting like in the previously explained algorithm, and used to build a relevance weighting function. Two different approaches to build that function have been evaluated (plus a third one which consists in no weighting at all). Results for each of them will be analyzed in V.3.

- **Method 1:** The function used is a decreasing exponential (Eq. 31), a plot is given at Fig. 34. Here, d is the component to component input distance, mP is the mean of all the distances and η is a parameter which adjusts how fast farther components weights are reduced. In this work η=0.3 seemed and appropriate value. The concept is simple: trajectory components having more similarity to those of the opposing video receive higher weights, thus enforcing the concept of focusing on similar motion.

$$w_{M1}(d) = e^{d*\ln(\eta)/mP} \qquad \text{Eq. 31}$$

- **Method 2:** The function used follows an exponential law, but with an upper bound (Eq. 32), a plot can be seen in Fig. 34 (left). It assigns value η to the mP mean and α to zero. In general, this function will weight upper distanced values higher, that is certainly not intuitive, there is however a good reason to do so.

$$w_{M2}(d) = min\left\{e^{\frac{d}{mP}*ln\frac{\eta}{mP}}, 1\right\} \quad \text{Eq. 32}$$

Let's suppose we first compare two videos with different motion in general, most of the component distances obtained will be high, but some of them could still be low (noisy trajectories or secondary motion). In that case the mean of the component distances would be high and thus the few distance components with associated low distances would receive a low weight, reducing their relevance. This way the final result is biased towards the higher distanced components. This is useful because almost all videos share a number of similar components, the reason is they might be a number of short, noisy and hence "false" trajectories that may fit well enough legit trajectories in the comparing video. These false trajectories will then erroneously bias the metric towards lower distances and we want to avoid this.

In the opposite case, two videos with similar motion would have mostly low component distances, hence the mean of these would be close to these "good" results. Here lower 'legit' distances would receive a penalty in terms of relevance as well, but being closer to the mean distance they would not be much affected as in the opposite case. High distances corresponding to the few dissimilar motions would certainly be emphasized, but now the upper bound limits how much their weight is increased.

Overall using this weighting technique all the videos receive a distance penalty, but this penalty is higher for more dissimilar videos, which in the end increases contrast and facilitates posterior video clustering.

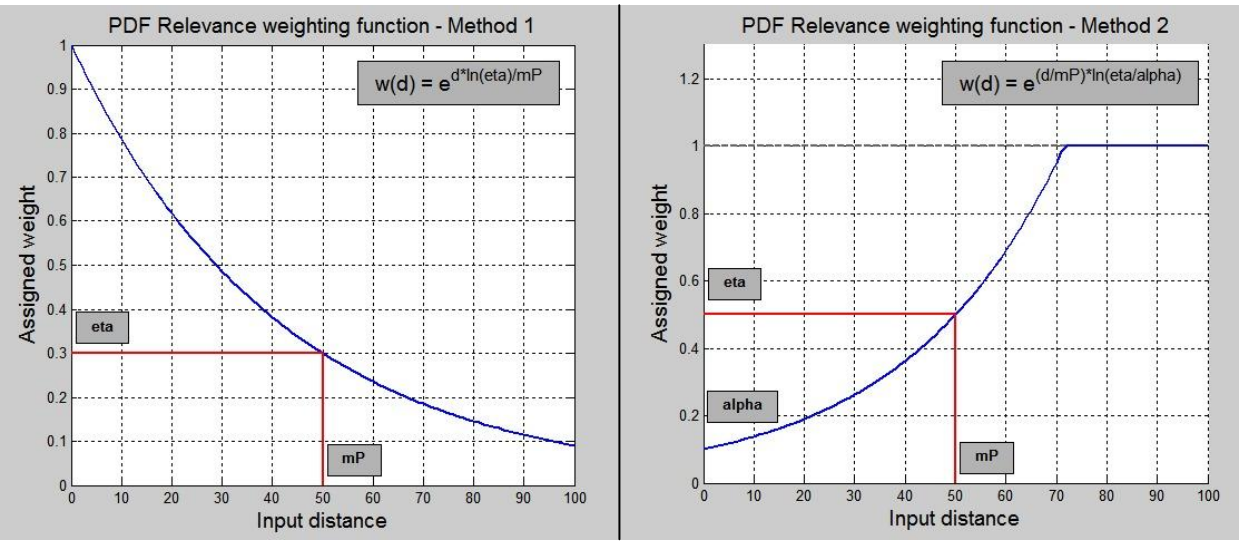


Fig. 34. PDF second pass weighting functions. Each function corresponds to one of the methods tested.

At each video to video comparison this process is run in each direction (video 1 to video 2 and vice versa), the output is two weighting vectors of lengths equal to the number of components of each video. These weighting vectors are used in the second pass to reweight the data matrices when computing the global PDFs. Remember that the process is done for each signature type independently, hence four times for every video to video distance comparison.

This additional weighting stage adds the “Tri-Weighted” to the name of the algorithm, remember that in this algorithm data is finally weighted using three factors: component lengths, trajectory originating objects sizes, and as explained here, cross-relevance.

IV.4.2 Second pass: final distance calculation

The new cross-relevance weighting matrices are combined with the previously obtained length/size weighting matrices using the dot product. The resulting global weighting matrices are used to estimate the two global distance PDFs.

Unfortunately, the previous weighting function breaks linearity, resulting in two different estimates. The reason is the weighting matrices for each video are now different. The resulting PDFs are very similar, but applying the final distance criteria used in the method explained in IV.3 returns slightly different results. Because of this the process has to be done for the two “branches” (from video 1 to video 2, and then for video 2 to video 1, see Fig. 36), note than in the “single pass” algorithm this is unnecessary as the weights matrices are the same for both branches.

To solve this issue both estimates are combined. This is straightforward because the estimation points used to calculate the PDF depend solely on the input data, which is the same for the two

branches; hence no interpolation is needed to compute the combination of the two PDFs: simple sample by sample averaging is possible.

Finally, the distance between the two videos is extracted using this combined PDF in the same way as in the previous method: searching the first maximum that exceeds the given threshold.

This two pass method renders more accurate results. However its computational cost is much higher. Compared to the single pass approach, it needs a considerable number of PDF estimates while the previous one needs just one.

Fig. 36 shows the complete workflow for the double pass method. Fig. 35 shows individual PDFs used to compute relevance weights along with the final global PDF.

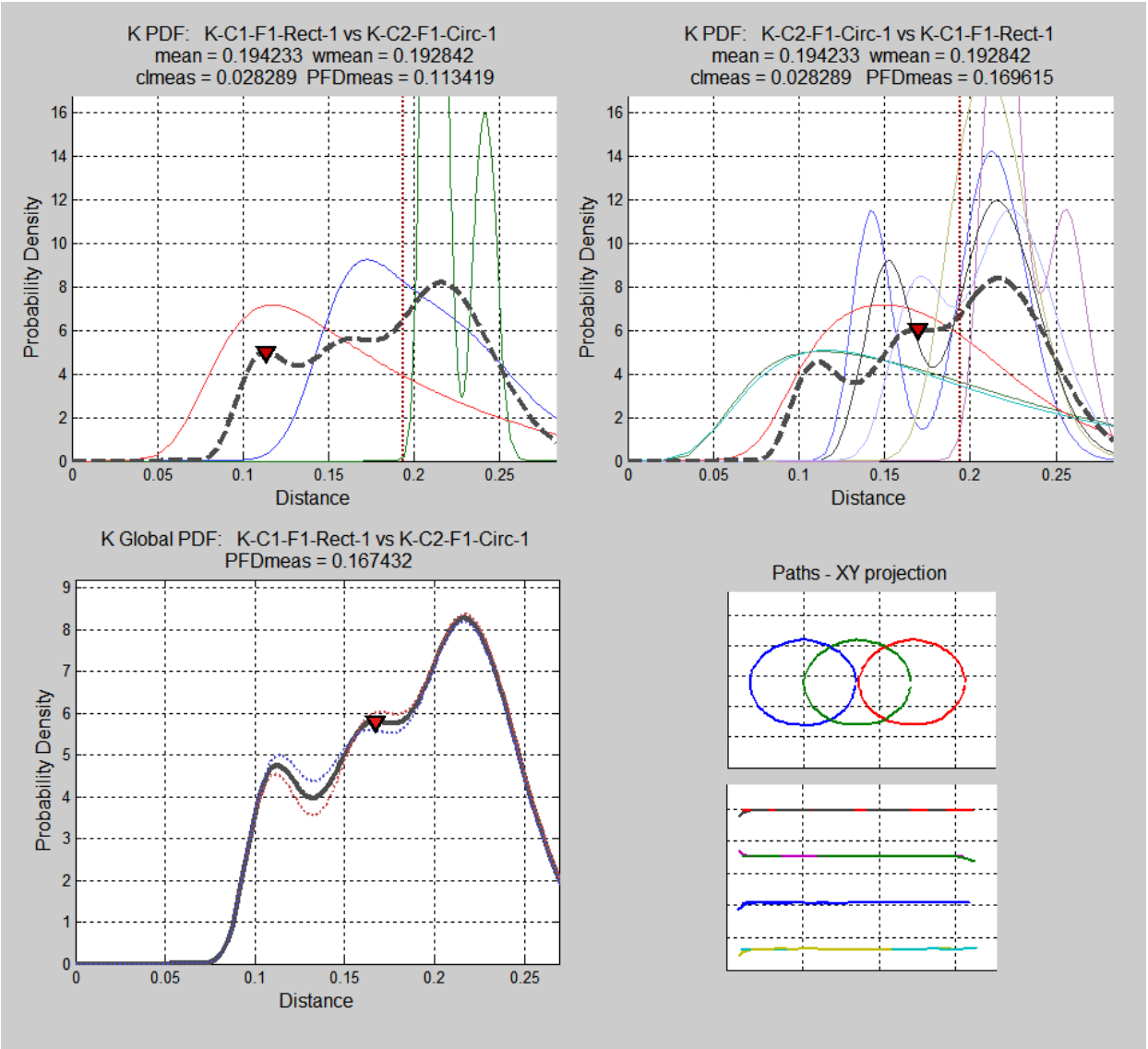


Fig. 35 Final algorithm PDF distances examples for curvature signature. Up: individual component PDFs (color) and global PDF estimation (dashed gray) for the two branches; calculated distance (local maximum) appears as a red triangle. Down-left: global PDFs for each branch (red/blue) and final averaged PDF (gray). Down-right: trajectories of the videos involved.

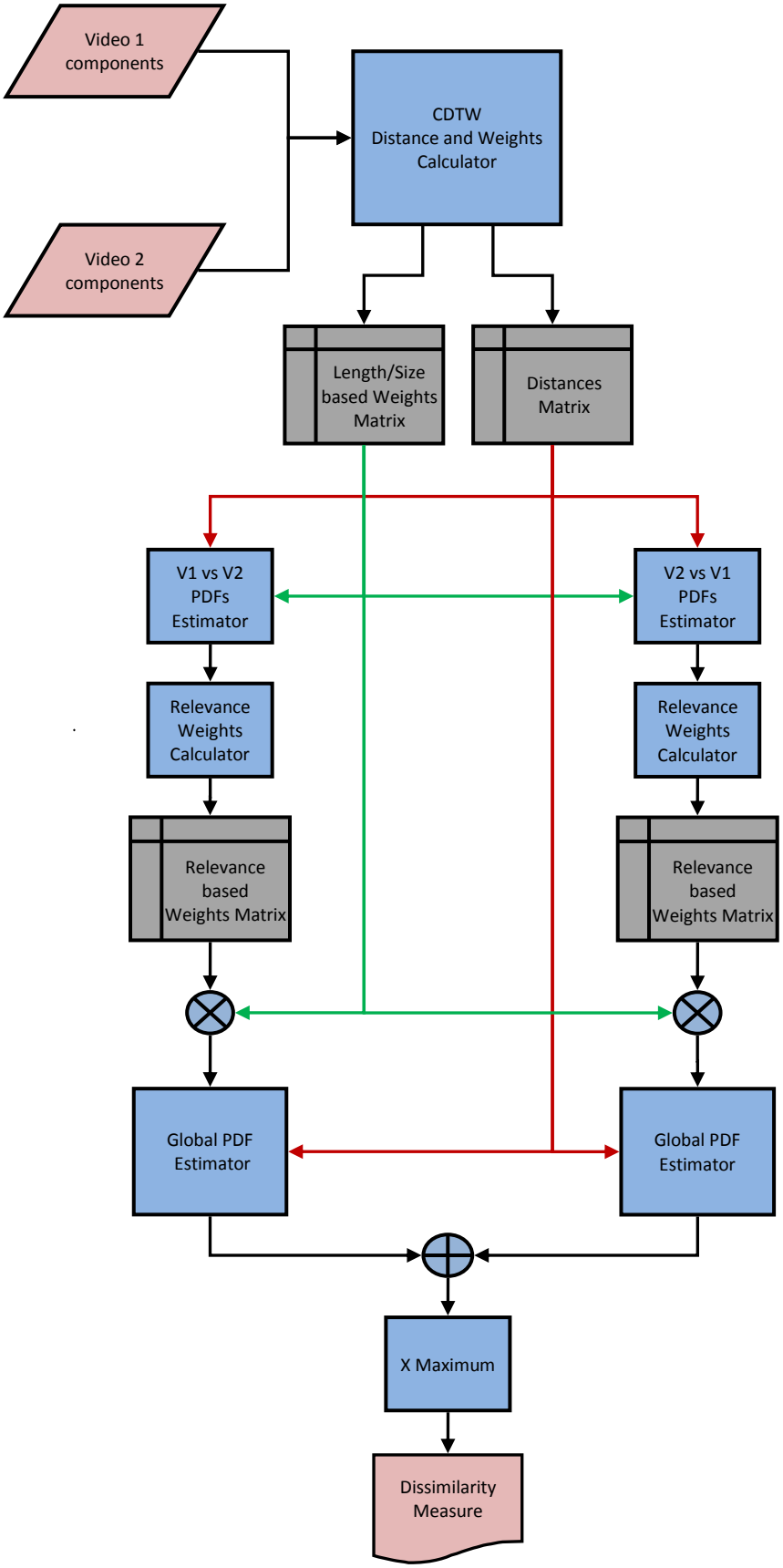


Fig. 36 Similarity Measure Workflow. Note this is for only two videos and one signature. The same process is done four times to compute four dissimilarity measures, one for each signature type.

V Hierarchical clustering: Experimental results

In this chapter, experimental results are detailed for our final "Double Pass Tri-Weighted PDF" algorithm. Also these are confronted to three other video similarity metrics found in the literature, one based on per frame similarity, another based on color sets, and a third one based on motion.

Agglomerative hierarchical clustering is conducted to verify the quality of the metric proposed in this work. As comparing two sets of dendrograms is not obvious a scoring framework is described in V.1. The alternative algorithms benchmarked here are explained in V.2. Comparative results between all the tested metrics is provided in V.3. Specific results for the other video similarity metrics are provided in V.4.

V.1 Scoring framework

In order to check whether the dissimilarity measure works properly, a clustering is applied to 9 sets of videos taken from the database described in II.2. The video to video distances are calculated using our algorithm, four set of matrices are obtained, one for each signature type. As each signature has a different range of distances these are normalized prior to combining them into a single distance matrix that is used for the actual clustering.

The results obtained are hierarchical cluster trees, a method has been defined to score how well the analyzed videos cluster together into their natural "families".

For each video, the nodes where it gets grouped to videos of its same family are searched. This leads to one, two or three nodes, depending on how it got clustered. At each of these nodes, the ratio of videos from the same family to the total of videos is calculated, this gives a score for that video at that given node. The result for that video is then weighted across all the considered nodes, giving higher weights to the firsts nodes found, this reduces the impact of one single miss-clustered videos over the result of the other three correctly clustered videos. This way a score for each video is obtained. The global raw score is the average of these results.

We want the score to reflect a clustering quality index. The raw score range using this method is [0.085 1], the point is expanding such range to [0 100], scoring 0 for what would be random clustering and 100 to perfect clustering. We ran 10^5 scoring iterations using random matrices with uniform distribution, the result was 0.2113; this value is taken as the expectancy of a random distance matrix clustering score. Score normalization (Eq. 33) is carried to fulfill the desired scoring behavior.

$$Score_{NORM} = \frac{100}{1 - E\{Score_{RAND}\}} Score_{RAW} - \frac{100 * E\{Score_{RAND}\}}{100 - E\{Score_{RAND}\}} \quad \text{Eq. 33}$$

Besides from final scores, pseudo confusion matrices are presented here. These are not to be mistaken with supervised learning confusion matrices, these matrices show how each family of videos tends to cluster to the other families, being the rows the input video families and columns the output clustering "probability", hence the sum of the rows equals 1.

V.2 Alternative algorithms

Three similarity metric algorithms have been selected, implemented and tested against our approach. These are described in this section.

V.2.1 SSIM based clustering

Structural similarity index is a full reference metric originally proposed to measure image quality after compression [42]. It compares a compressed image to its original peer and establishes a similarity measure based on HVS rather than more typical metrics like PSNR and MSE, which are less consistent with human eye perception.

SSIM is conducted on the comparison of three signals. The means, variances and covariance are respectively used to compute comparisons of luminance, contrast and structure. A more intuitive representation is given at Fig. 37: luminance is obtained directly as the mean of the signals, contrast is computed after subtracting it to the signal, finally contrast is used to normalize the signal and a structure measure is extracted. When all three components are equally weighted in the final SSIM measure, the result can be written as in Eq. 34.

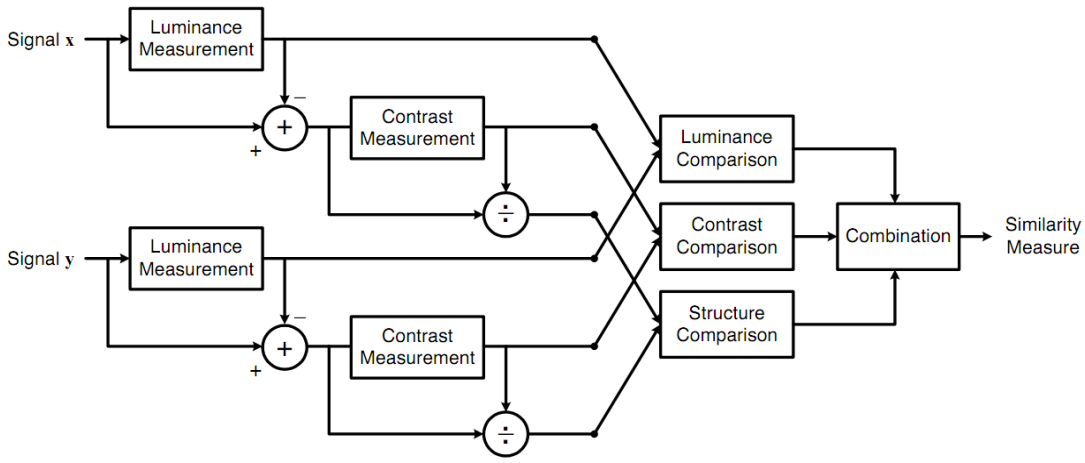


Fig. 37. Diagram of the Structural Similarity Index Measurement

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad \text{Eq. 34}$$

Images statistics use to be highly spatially non stationary and distortions are space variant. Because of this the SSIM index is not used directly on the whole image, instead of this the image is divided into a partition of blocks and SSIM indices are calculated for each one of them, a map of SSIM indexes is built and the final similarity index is calculated as the mean of these similarity indices (Eq. 35):

$$MSSIM(X,Y) = \frac{1}{M}\sum_{j=1}^{M} SSIM(x_j, y_j) \quad \text{Eq. 35}$$

In order to test this metric on videos, a multi frame comparison is done. From each video a fixed number N of frames is taken at equally spaced intervals. The extracted frames are compared one to one using the MSSIM metric and the results are averaged.

V.2.2 Motion Texture clustering

Motion Texture was proposed by **Yu-Fei Ma et al.** [43]. Basically it transforms the MVF into a number of directional slices of energy, a set of moments are measured on these slices. The result is a multi-dimensional vector, called Motion Texture is formed, which is the base for a similarity metric.

In the MVF, let (i,j) be the position of macroblocks in raster scan order, and $V_{i,j}(\Delta x_{i,j}, \Delta y_{i,j})$ be the motion vector of macroblock $MB_{i,j}$. Energy $En_{i,j}$ in macroblock $MB_{i,j}$ is defined as in Eq. 36:

$$En_{i,j} = \sqrt{\Delta x_{i,j}^2 + \Delta y_{i,j}^2} \quad \text{Eq. 36}$$

The energy in MVF is mapped to a unit circle, rectangular coordinates are constructed at the center of MVF, and polar coordinates at the center of unit circle. The process of mapping the energy in an MVF to a unit circle can be defined as in Eq. 37:

$$g(\rho,\theta) = \sum_{x_i=-w}^{w}\sum_{y_j=-h}^{h} En_{i,j} \qquad if \begin{cases} \rho = \overline{r_{i,J}} \\ \theta = \alpha_{i.j} \end{cases} \quad \text{Eq. 37}$$

Where $g(\rho,\theta)$ is the energy distribution function of unit circle, $\overline{r_{i,j}} = \sqrt{x_{i,j}^2 + xy_{i,j}^2}/\sqrt{w^2+h^2}$ is the normalized distance for macroblock $MB_{i,j}$ to the center of MVF and $\alpha_{i,j} \in [0,2\pi]$ is the orientation of motion vector $V(i,j)$. This process is called Circular Mapping, and the mapped unit circle is called Energy Unit Circle (EUC). In EUC, both object motion and camera motion present distinctive patterns.

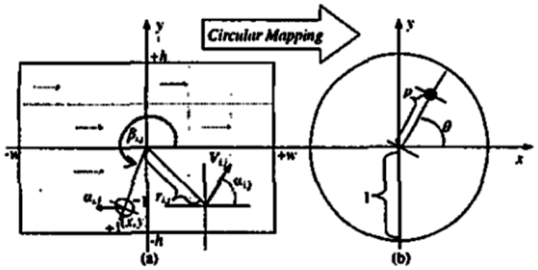


Fig. 38. Circular Mapping

In order to capture the temporal pattern of motion during a period of time, slices from successive EUCs along temporal axis are extracted. EUC is divided into four equiangular opposite sectors. Then, the energy in each sector is accumulated to the central lines along homo-centric circumference. Finally, directional slices from EUC volume are extracted at those central lines. This process is called directional slicing.

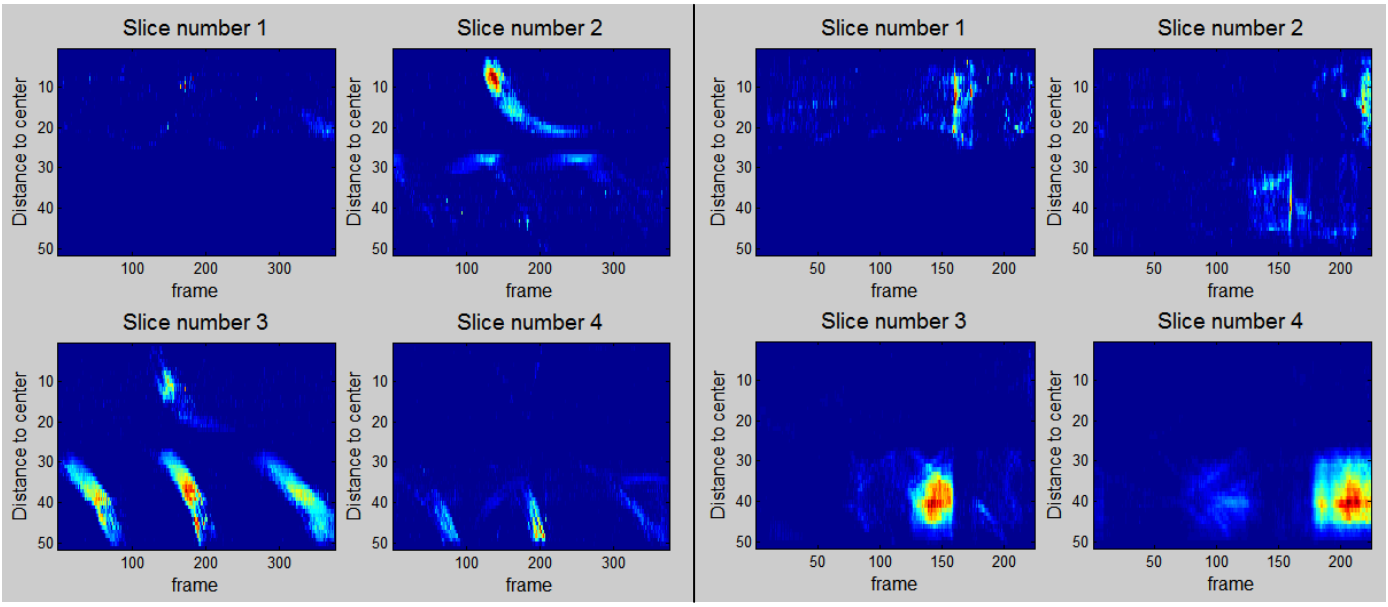


Fig. 39. Examples of slices. Left slices are from Road-06 video while right ones correspond to Foot-12.

To characterize slice images, a set of moments are calculated, assuming those slices have the size of MxN, the moments can be computed as in Eq. 38:

$$m_{pq} = \sum_{y=1}^{N}\sum_{x=1}^{M} x^p y^q f(x,y) \qquad \text{Eq. 38}$$

Where $(p,q) = \{(0,0),(1,0),(0,1),(2,0),(0,2),(3,0),(0,3)\}$and $f(x,y)$ is the energy distribution of a slice. Based on these moments, 9 values with specific physical meanings are calculated, these include center of mass, radii of gyrations, skewness and kurtosis. These values form a vector for each slice, hence for each video a 9x4 dimension feature vector T is obtained:

$$T = \{T^0, T^2, T^3, T^4\} \quad \text{Eq. 39}$$

$$T^n = \left\{m_{00}^n, \overline{COM_x}^n, \overline{COM_y}^n, \overline{ROG_x}^n, \overline{ROG_y}^n, Sk_x^n, Sk_y^n, K_x^n, K_y^n\right\} \quad \text{Eq. 40}$$

Since the dynamic range of each component of motion is quite different, normalization is needed when we compare two motion texture vectors. Assuming we have a video clip database, the motion texture is extracted from each clip. Then, each component of vectors is normalized by the inverse of the standard variance. Euclidean distance is adopted as similarity measure using the normalization coefficients as weights, the similarity measure then can be written as in Eq. 41:

$$Sim(T^a, T^b) = \sqrt{\sum_{k=1}^{4*9} \frac{\left(v_k^a - v_k^b\right)^2}{\sigma_k^2}} \quad \text{Eq. 41}$$

V.2.3 ICC (Image Characteristic Code) clustering

A fast video similarity search was proposed by **Cheung, S.-S** [44], it uses a similarity measure based on low level features extraction. Image characteristic code (ICC) is a joint feature representation made up of three statistical integers of every pixel components: Y, Cb and Cr. Means of these are calculated for every frame as in Eq. 42, creating a "Video Component".

$$\begin{aligned} m_1 &= \sum_{i=1}^{M}\sum_{j=1}^{N} Y_{ij} / MN \\ m_2 &= \sum_{i=1}^{M}\sum_{j=1}^{N} Cb_{ij} / MN \\ m_3 &= \sum_{i=1}^{M}\sum_{j=1}^{N} Cr_{ij} / MN \end{aligned} \quad \text{Eq. 42}$$

The Video Component (VC) is the set of image frames with the same ICC, which can be identified as a statistical feature cluster based on STD. According to the statistics of video component, the video similarity measure can be defined as follows: giving two video clips whose video component set is C_X, C_Y respectively, n_k is the number of common video component clusters in their intersection set. The video similarity D is the sum of common component clusters. In this way, the video similarity is measured by fine computation of video component based on the statistics of spatial-temporal distribution.

$$S(X,Y) = \sum_{k \in C_X \cap C_Y} n_k \qquad \text{Eq. 43}$$

This metric is used for fast video search and it is focused mainly to give high discrimination ratio rather than precise distance measures. This works good on long videos (30 minutes or more), that were the original target of the algorithm), but on short videos there are many chances that many distances computations return zero similarity. A simple tweak has been done to make it more efficient at clustering videos like in the database used in this work. More precisely when computing the distance of two videos, at each video component comparison the "binary" intersecting criterion is not used. Instead, all sets are considered as intersecting, but the added similarity is weighted by the distance of the comparing sets. Given two videos X and Y, with their video components set C_X and C_Y, the similarity metric would be as in Eq. 44

$$S(X,Y) = \frac{\sum_{i=1}^{\overline{\overline{C_X}}} \sum_{j=1}^{\overline{\overline{C_Y}}} \left(n_{Xi} + n_{Yj}\right) * e^{-k\left\|\overrightarrow{C_{Xi}} - \overrightarrow{C_{Xj}}\right\|}}{\sum_{i=1}^{\overline{\overline{C_X}}} n_{Xi} + \sum_{j=1}^{\overline{\overline{C_Y}}} n_{Yi}} \qquad \text{Eq. 44}$$

The dissimilarity metric is taken by subtracting this measure to one, as in Eq. 45.

$$D(X,Y) = 1 - S(X,Y) \qquad \text{Eq. 45}$$

V.3 Proposed algorithm results and benchmark

In this subsection the three PDF weighting strategies and normalization methods are evaluated. Using the best choice of these ones, global results for our algorithm are presented and analyzed. Then the results of all the metrics tested are confronted. Finally the specific results for the third party metrics are presented and commented.

V.3.1 Analysis for PDF weighting strategies and normalization methods

The video sets have been processed using all four possible normalizations and the three PDF "second pass" weighting methods. Results can be seen at Fig. 40.

In terms of normalizations, it is clear that no normalizing gives very poor results. Trajectory level normalization described in III.4.1 improves results slightly, but even in the best case it only improves 8.8 points, what is rather poor. The mean/variance and the power normalizations give much better results, when combined with the second weighting method the scores are pretty high, thus discerning motion between families with good accuracy. With the sets of videos used in this test, Power normalization yields the highest score at 72.1 points, however the mean/variance one is really close. In fact, if we look at per Video Set results in Fig. 42, we can see that the winning method depends on the actual selection of videos.

In terms of weighting methods, the second one is a clear winner. Surprisingly the first method tends to score even worse than the third method which simply does not reweight the data.

Scores	Method 1	Method 2	Method 3
None	11,4	21,4	11,0
Zero Mean Unit variance	24,6	66,3	41,9
Power	24,9	72,1	37,1
Trajectories	20,2	23,1	13,0

Fig. 40. Normalizations vs. PDF weighting methods. Best combination is highlighted in blue.

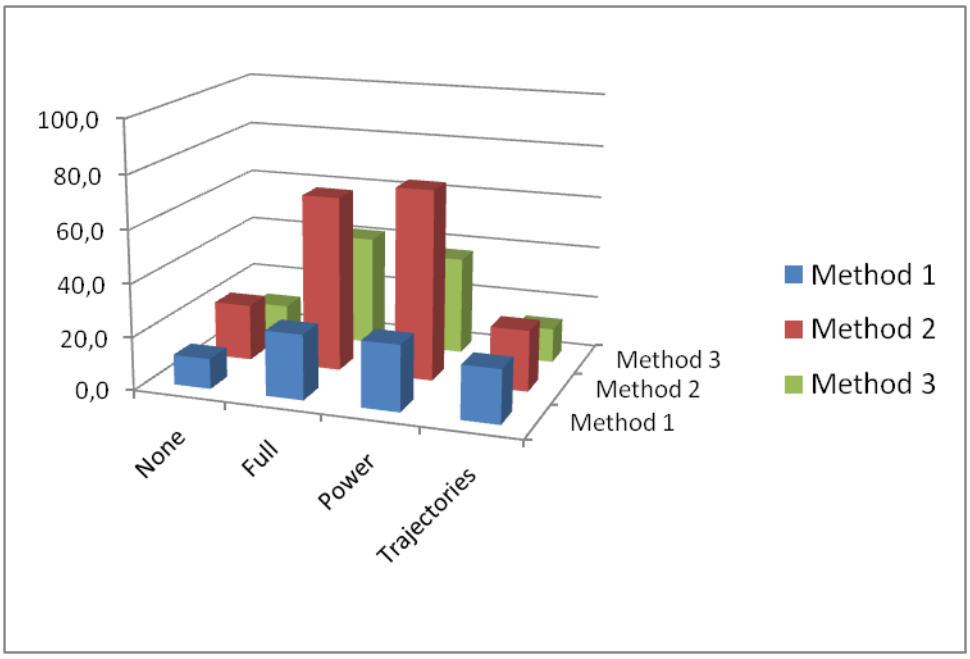


Fig. 41. Normalizations and PDF weighting methods scores.

Scores	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 9**	**Mean**
Mean/Variance	70,3	65,4	63,5	100,0	62,2	39,8	79,7	54,3	61,9	66,3
Power	66,1	84,7	72,5	92,6	66,5	57,5	64,8	80,1	63,8	72,1

Fig. 42. Per Video Set results for Power and Mean/Variance normalizations using the best weighting method. Winning results are highlited in blue for each Video Set.

From now onwards, the results considered for benchmarking against the rest of the algorithms will be those obtained with the “method 2 weighting” and Power component normalization.

Per family results for all the algorithms can be seen in subsection V.3.3 at Fig. 47, also individual family improvements are given in Fig. 47.

V.3.2 Proposed algorithm results with best method and normalization.

Fig. 43. shows the results and pseudo confusion matrices for the presented algorithm. Overall quality score is 72.1 points.

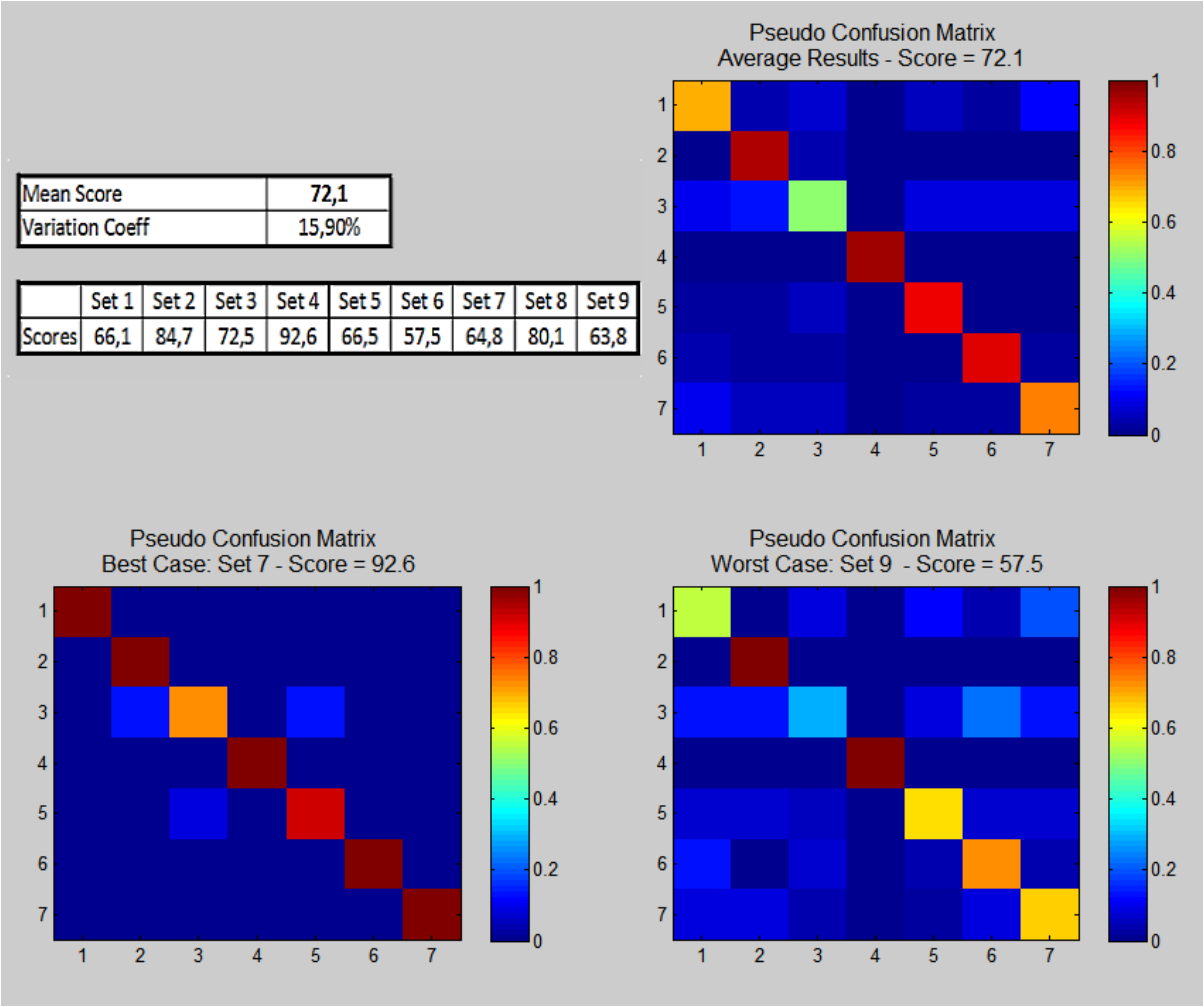


Fig. 43. Proposed algorithm Results

As can be seen in the confusion matrices (Fig. 43), some families have problems clustering together. Some particular cases are detailed here.

The “Road Surveillance” family (number 3) has some videos that do not cluster well with the rest in the same class, one good example is “Road 9”. As can be seen in Fig. 44, this video has faulty trajectories, in part because of erroneous camera motion estimation: there is little camera motion, but the huge area that covers the cars moving with similar motion has fooled the estimator. This and the occlusion of cars returned noisy trajectories. These trajectories compared to the ones of “Road 17” video, which are very well defined, gives high distances between these two videos. At the same time, “Halt 04” video also has noisy trajectories, in this case because of the low quality and low resolution

(320x240) of the original video source. Hence trajectories of those “Halt 04” and “Road 9” videos are statistically relatively close, so distances between these two videos is lower than desired.

“Bill 11” video is added as well to illustrate how some families have similar motion: billiard videos have many trajectories with very low curvature (it is in fact rectilinear motion) and decreasing speed, which are not very different from those of the “Road” class were some trajectories are also rectilinear with decreasing or increasing speeds, mainly because of the perspective, that is also similar for both videos.

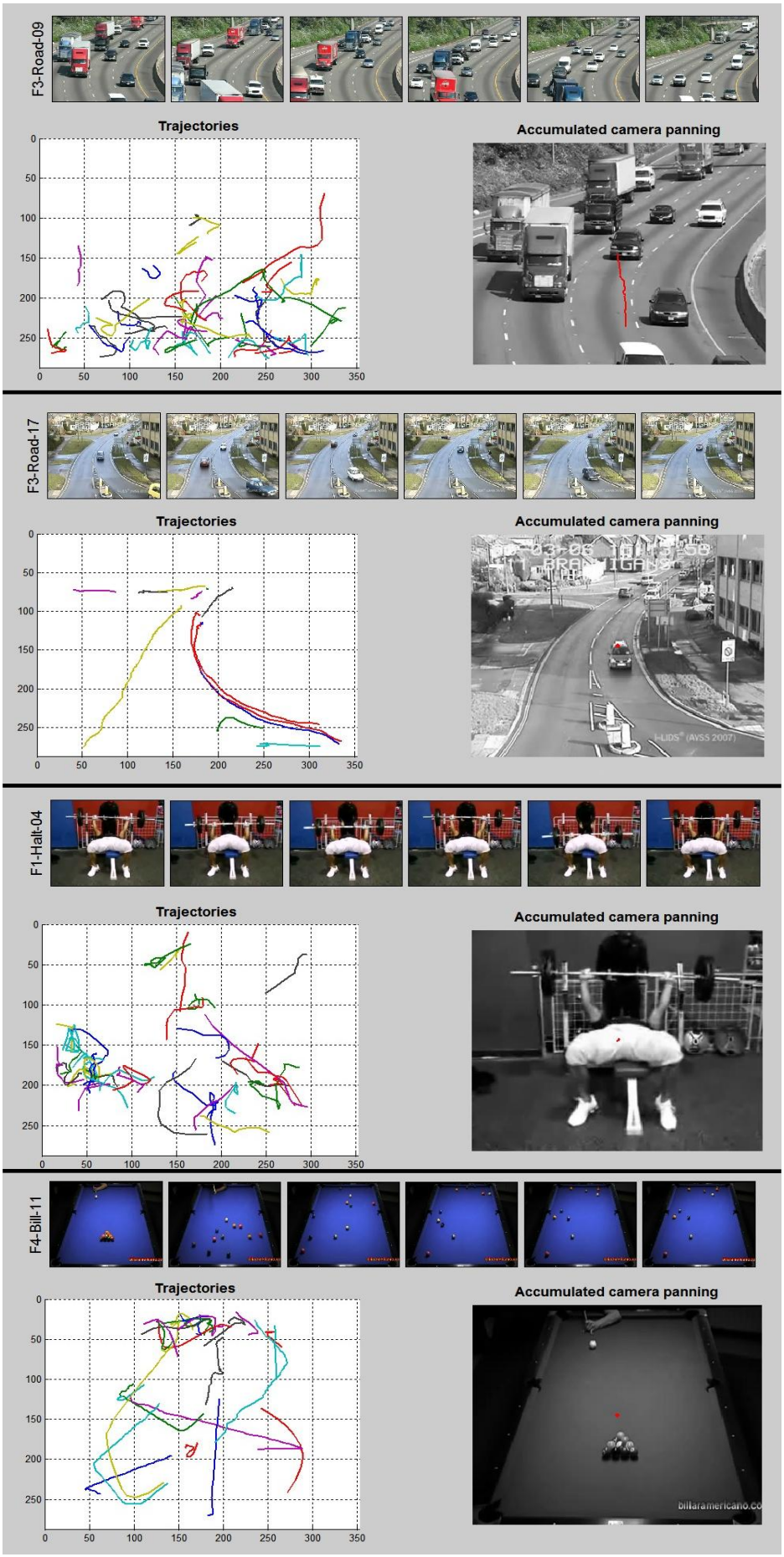


Fig. 44. Trajectory examples picked up to explain miss clustering.

V.3.3 Comparative results against other algorithms

Fig. 45 shows a summary of all the algorithms average clustering results. Coefficient of variation has been added because it gives a better idea of reproducibility than the standard deviation as results must be understood in the context of the average of the analyzed data. Raw and percentage increase of the proposed algorithm over the rest are provided as well.

It can be seen that our approach outperforms those basic metrics, providing 28.1 points increase over the ICC ones (second best). Also the lower coefficient of variation indicates the results are less dependent of the selection of videos.

	Proposed	ICC	MT	SSIM
Mean Score	72,9	44,8	39,1	32,6
Variation Coeff.	22,9%	32,8%	37,5%	59,9%
Increase (raw)	0	28,1	33,8	40,3
Increase %	0%	62,7%	86,4%	123,6%

Fig. 45. General scores against other algorithms

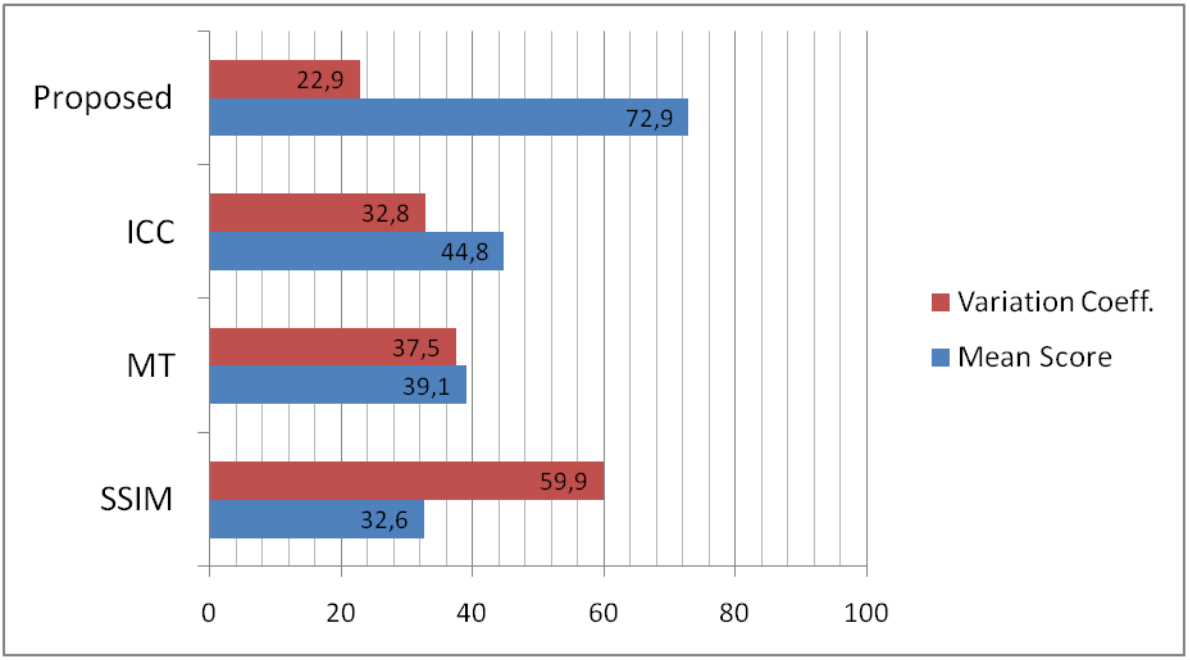


Fig. 46. General Scores

Fig. 47 shows scores for each video family and algorithm. Excepting for Pool Diving and Football videos at which our proposal struggles to cluster them, the results are overall better.

	Weightlifting	Pool Diving	Road surveillance	Billiards	Football	Casino	Group Dance
Proposed	55,2	91,3	32,1	94,8	82,3	85,7	63,1
ICC	16,9	3,7	54,3	75,4	92,4	45,3	25,6
MT	32,3	66,8	23,9	55,0	28,1	63,3	17,3
SSIM	16,8	19,5	26,5	78,2	52,3	12,6	22,1

Fig. 47. Scores for each video family

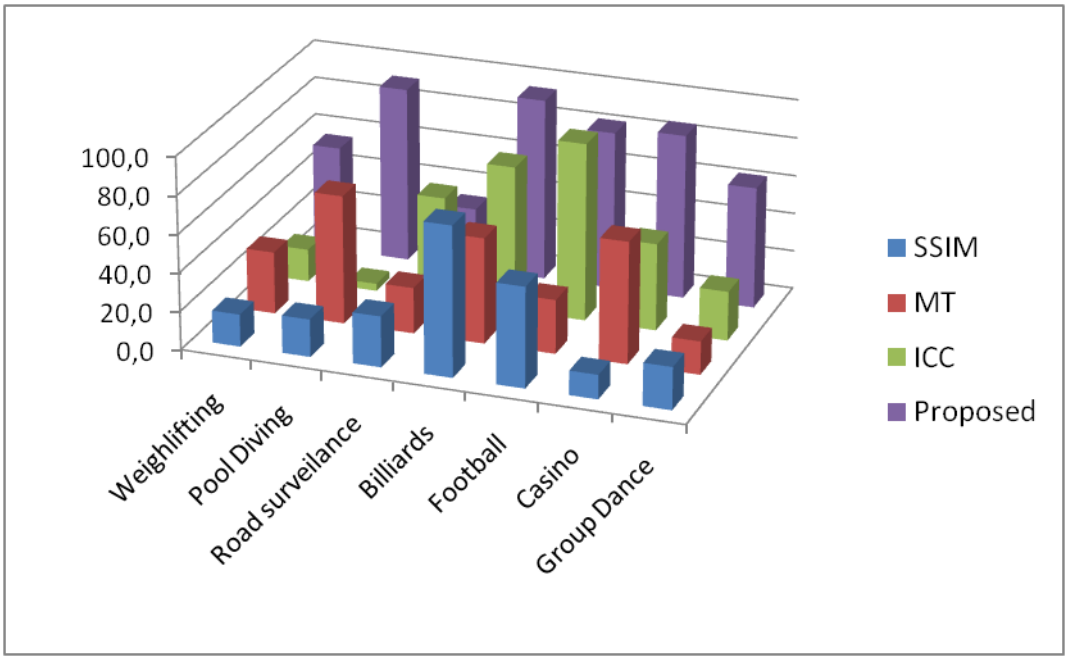


Fig. 48. Per video family scores

V.4 Specific results of competing metrics

In this subsection, specific results and pseudo confusion matrices for the three competing algorithms are provided. Also interpretations of the results in terms of video content are formulated.

V.4.1 SSIM

The SSIM extended to video does work in some way; however the score obtained his low at 32.6 points. By comparative observation of best and worst cases it can be seen that best scores are biased, the reason is many videos in the database come from the same source as can be seen in Fig. 50. (like the weightlifting competition), in these cases the frames share the same background that is usually still. This situation boosts the similarity measures between those videos' frames. The opposite case can be seen for Set 6, were all the videos come from different sources, here the effectiveness of the algorithm is severely hampered and a rather disappointing score of 9.4 points is obtained. Overall the coefficient of variation is 59.9%, the highest of the tested methods.

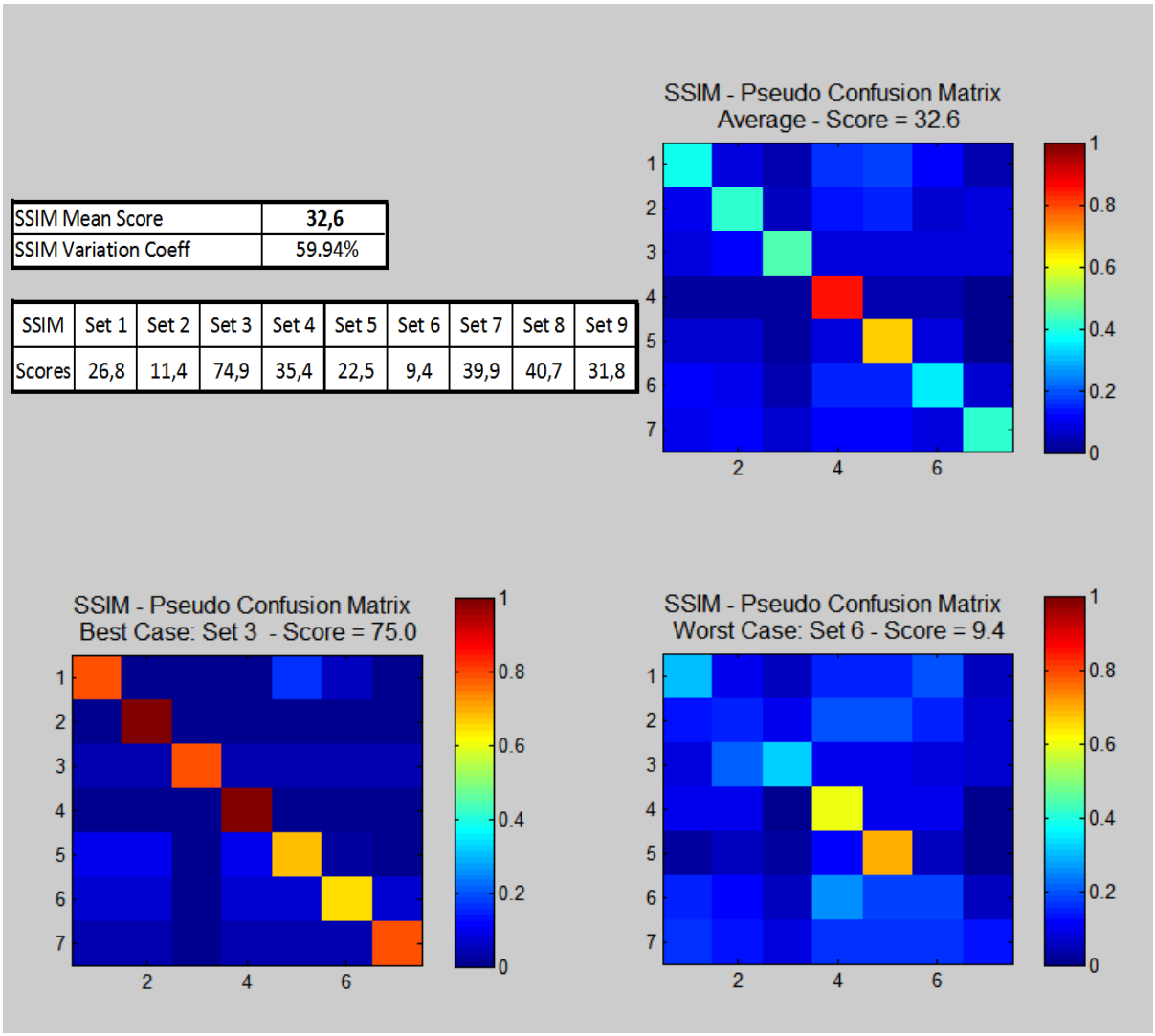
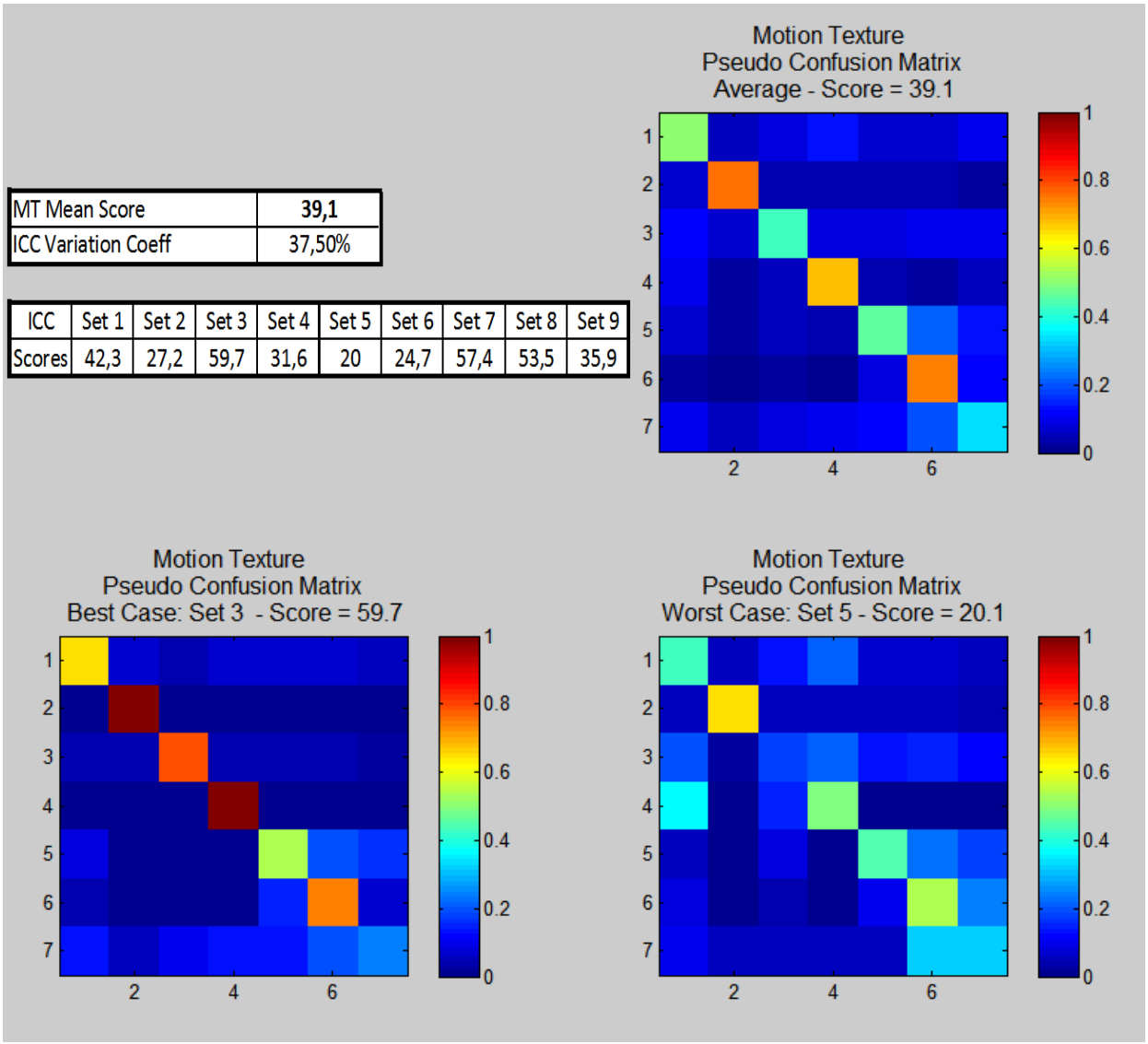


Fig. 49. SSIM results

Fig. 50. Best (Set 3) and worst (Set 6) clustering cases for SSIM algorithm. Source variety in the worst case explains worse results while videos with same background benefit the algorithm.

V.4.2 Motion Texture

Motion Texture clustering gave better results than the simpler SSIM with a score of 39,1 points. This algorithm has been tested with both original MV and camera motion compensated MV, final results were similar with a slight advantage of the non corrected MV implementation. Despite having a lower score than our algorithm, it can be seen as the pseudo confusion matrices share some similarities, especially families 2 (pool) and 4 (billiard) are again the ones that cluster the better, while family 3 (road) gets the lower scores. Certainly this is not a coincidence: considering that even in both algorithms are based on motion, they still are very different in nature, so the fact they return similar results show that family 2 and family 4 have more distinguishable motion compared to the rest.



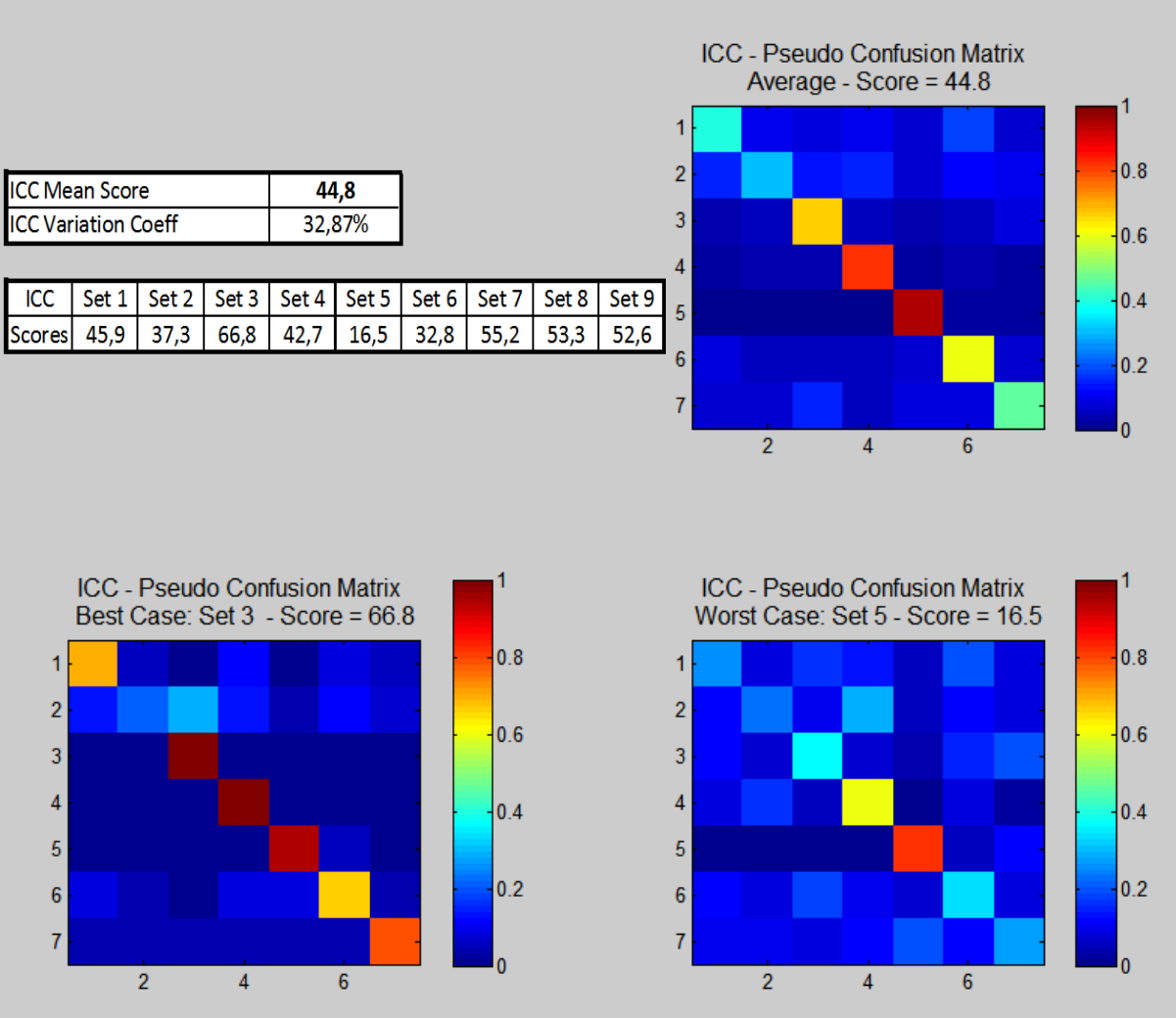
MT Mean Score	39,1
ICC Variation Coeff	37,50%

ICC	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9
Scores	42,3	27,2	59,7	31,6	20	24,7	57,4	53,5	35,9

Fig. 51. Motion Texture scores

V.4.3 ICC (Image Characteristic Code)

ICC clustering gave decent results, with an average score of 44.8 points, being thus a clearly better distance metric than the SSIM index and having an edge over MT. That said, like happened with SSIM the best results are also biased by the same reason, but the gap between best and worse results is lower, which is confirmed by a lower coefficient of variation of 37.5%.



ICC Mean Score	**44,8**
ICC Variation Coeff	32,87%

ICC	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9
Scores	45,9	37,3	66,8	42,7	16,5	32,8	55,2	53,3	52,6

Fig. 52. ICC results

With this algorithm reasons for miss clustering can be deduced easily. In the average pseudo confusion matrix, for set 5 (Fig. 52) it can be seen as family 2 (diving) and family 4 (snooker) videos tend to cluster together. The reason is both families share a major component of blue color (excepting one video from family 4), as this algorithm seeks for similar color, the distances between videos of those families were low.. At the same time, the best clustering family is number 5 (soccer), because most of the images are dominated by a green component from the grass.

Fig. 53. Best (Set 3) and worst (Set 5) clustering cases for ICC algorithm. Again, source repetition benefits clustering, here color similarities are the key point for correct clustering.

VI Conclusions

A novel metric for videos which relies solely on motion has been successfully implemented and tested. The results proved it outperforms a selection of algorithms based on color, image structure and motion. Also, it fulfills the goal of analyzing motion with invariance to precise position, scales, frame rates, some compression distortion and most processing that videos usually suffer.

VI.1 Novel contributions

Many metrics that deal with video motion have been proposed in the last years for video retrieval. Some classification or indexing proposals make use of trajectories, but mostly for specific object activity detection or just trajectory retrieval, for example **F. Bashir et al.** [45] used trajectories to classify words from the Australian Sign Language and specific human actions. Here, we tried to generalize the use of trajectories to measure distances between videos independently of specific video content.

To do such we first chose to work on trajectory signatures and we presented a novel framework to measure distance between two trajectory signatures. First we introduced a new dynamic time stretching algorithm that combines DTW and DDTW while successfully weighting their influence independently of the input sequences statistics, second we up-scaled this algorithm to find the best scale and offset between input sequences for overall minimum distance.

To process the high number of distances between two videos trajectory components we built a new PDF based metric, which uses a double pass scheme to weight the data and extract a final distance that indicates whether videos share similar motion.

Such approach has two main strong points. First, the MS-CDTW metric allows some flexibility at seeking for similar trajectories, but maintaining global shape. Second, the PDF based metric permits assigning low distances to videos that share enough motion similitude while obviating the presence of a certain amount of secondary dissimilar trajectories, which include fake noisy ones.

VI.2 Criticisms

In terms of metric accuracy, our algorithm certainly outperforms the other metrics tested in this work. That said, our algorithm focus on giving low distances to videos that have similar trajectories, and the videos selected for this test were handpicked using this visual criterion. At the same time, the other algorithms are based around different features, so our algorithm has an edge here given by a “biased” selection of testing data. Further testing should be conducted using a more variable video database. Another limitation related to videos used is related to the length of these: our algorithm works well

with short videos (up to 30 seconds), but with very long videos the high number of different trajectories would fool the PDF distance stage as certainly enough good matches would occur even for videos of very distinct content. So, the longer the videos are, the more chances they will receive a low distance even if their trajectories tend to be dissimilar.

Another limitation is related to the speed of computing the results. Our algorithm should perform adequately in query-by-example video retrieval, however it has a huge drawback speed wise. Video retrieval can be divided in two main steps, offline and online. In the online part, the videos in the database are processed to extract a set of features. This is done only once, so computational efficiency is not crucial at this point. Online step is carried when a query is made; the query video features are extracted and then compared to those stored in a feature database. Because databases may be very large, computational efficiency is crucial in this step. In most algorithms, this final distance comparison is very fast, sometimes just a Euclidean distance is needed between the feature vectors previously extracted and the ones stores in the database. Unfortunately in our approach the line between offline and online processing has been pushed too far to the online side. More specifically, in this work, offline processing consist in the extraction of the trajectories and pre-processing them into normalized signature components, while online processing implies calculating all inter-component distances and analyzing those using PDFs to get final video distances. All those heavy computations in the online phase make this algorithm impractical even under wise code implementation and use of the most powerful servers.

Finally, the effectiveness of our algorithm relies much on the quality of the trajectory extraction, which is not an obvious task when using block matching techniques. This could be addressed with other techniques to extract trajectories, like object segmentation and tracking. But this would mean giving a more specific purpose to this works, and of the premises of this project is maintaining generality respect to the input videos' content.

VII References

1. **Kraaij, Alan F. Smeaton and Paul Over and Wessel.** Evaluation campaigns and TRECVid. *MIR'06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval.* s.l. : ACM Press, 2006, pp. 321--330.

2. *Content based video indexing and retrieval.* **Smoliar, S.W. and Zhang, HongJiang.** 2, s.l. : Multimedia, IEEE, 1994, Vol. 1, pp. 62-72.

3. *Integrating visual, audio and text analysis for news video.* **Qi, Wei, et al.** 2000. International Conference on Image Processing. Vol. 3, pp. 520-523.

4. *Content-based classification, search, and retrieval of audio.* **E. Wold, T. Blum, D. Keislar, and J. Wheaton.** 3, 1996, Vol. 3, pp. 22-26.

5. *Audio feature extraction and analysis for scene segmentation and classification.* **Z. Liu, Y. Wang, and T. Chen.** 1, 1998, J. VLSI Signal Process. Syst., Vol. 20, pp. 61-79.

6. *A survey of MPEG-1 audio, video and semantic analysis techniques.* **U. Srinivasan, S. Pfeiffer, S. Nepal, M. Lee, L. Gu, and S. Barrass.** [ed.] Springer Netherlands. 1, 2005, Multimedia Tools and Applications, Vol. 27, pp. 105–141.

7. **L. Agnihotri, D. Nevenka.** Video Clustering Using SuperHistograms in Large Archives. *Advances in Visual Information Systems.* s.l. : Springer Berlin, 2000, Vol. 1929, pp. 255-268.

8. *Classification of summarized videos using hidden Markov models on compressed chromaticity signatures.* **C. Lu, M. S. Drew, and J. Au.** 2001. 9th ACM Int. Conf. Multimedia. pp. 479–482.

9. *On the use of computable features for film classification.* **Rasheed, Z., Sheikh, Y. and Shah, M.** 2005, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, pp. 52- 64.

10. *Video indexing using optical flow field.* **Ardizzone, E. and La Cascia, M.** 1996. International Conference on Image Processing,. Vol. 3, pp. 831-834.

11. *Video indexing using MPEG motion compensation vectors.* **Ardizzone, E., et al.** 1999. IEEE International Conference on Multimedia Computing and Systems. Vol. 2, pp. 725-729.

12. *Motion-based classification of cartoons.* **Roach, M., Mason, J.S. and Pawlewski, M.** 2001. International Symposium on Intelligent Multimedia, Video and Speech Processing. pp. 146-149.

13. *Using camera motion to identify types of American football plays.* **Lazarescu, M. and Venkatesh, S.** 2, s.l. : International Conference on Multimedia and Expo, 2003, Vols. Multimedia and Expo, 2003, p. 181.

14. *Motion Based Video Classification for SPRITE Generation.* **Deshpande, A.A. and Aygun, R.S.** 2009. International Workshop on Database and Expert Systems Application. pp. 231-235.

15. *Nonparametric motion characterization using causal probabilistic models for video indexing and retrieval.* **Fablet, R., Bouthemy, P. and Perez, P.** 4, s.l. : IEEE Transactions on Image Processing, 2002, Vol. 11, pp. 393-407.

16. **Emile Sahouria, Avideh Zakhor.** *A Trajectory Based Video Indexing System For Street Surveillance.* 1999.

17. *Real-Time Motion Trajectory-Based Indexing and Retrieval of Video Sequences.* **Faisal I. Bashir, Ashfaq A. Khokhar, Dan Schonfeld.** 1, s.l. : IEEE Trans. Multimedia, 2007, Vol. 9.

18. *Content based video matching using spatiotemporal volumes.* **Arslan Basharat, Yun Zhai, Mubarak Shah.** 2008, Computer Vision and Image Understanding.

19. *Fast Two-Step Half-Pixel Accuracy Motion Vector Prediction.* **K.H. Lee, J.H. Choi, B.K. Lee, and D.G. Kim.** s.l. : Electronics Letters, vol. 36, pp. 625-627, 2000, Electronics Letters, pp. vol. 36, pp. 625-627.

20. *A Comparative Study of Motion Estimation for Low Bit Rate Video Coding.* **He, C. Du and Y.** 3, s.l. : SPIE Proc., 2000, Vol. 4067.

21. *A new three-step search algorithm for block motion estimation.* **Li, Reoxiang, Zeng, Bing and Liou, M.L.** 4, Hong Kong : IEEE Transactions on Circuits and Systems for Video Technology, 1994, Vol. 4.

22. *Global motion estimation from coarsely sampled motion vector field and the applications.* **Su, Yeping, Sun, Ming-Ting and Hsu, V.** 2, 2005, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, pp. 232- 242.

23. *Encoding of arbitrary curves based on the chain code representation.* **Kaneko, T., Okudaira, M.** s.l. : IEEE Trans. on Communications, 1985, Vol. 33.

24. *Discovering Structure in Video.* **Dogan, Zafer.** s.l. : EPFL, 2010.

25. *Binary codes capable of correcting deletions, insertions, and reversals.* **Levenshtein, Vladimir.** 8, s.l. : Soviet Physics Doklady, 1966, Vol. 10.

26. *Optimum Uniform Piecewise Linear Approximation of Planar Curves.* **Dunham, James George.** 1, s.l. : Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1986, Vol. 8.

27. *Corner detection and curve representation using cubic B-Splines.* **Medioni, G., Yasumoto, Y.** s.l. : IEEE Int. Conf. on Robotics and Automation, 1986, Vol. 3.

28. *Principal component analysis in linear systems: Controllability, observability, and model reduction.* **Moore, B.** 1981 : IEEE Trans. on Automatic Control 26.

29. *Segmented trajectory based indexing and retrieval of video data.* **Bashir, F., Khokhar, A., Schonfeld, D.** s.l. : IEEE Int. Conf. on Image Processing, 2003. 2.

30. **Chang, William Chen and Shih-Fu.** *Motion Trajectory Matching of Video Objects.* s.l. : Dept. of Electrical Engineering, Columbia University, 2000.

31. *Determining the initial states in forward-backward filtering.* **Gustafsson, F.** 4, s.l. : Signal Processing, IEEE Transactions on, 1996, Vol. 44.

32. **Shandong Wu, Y. F. Li.** *Flexible signature descriptions for adaptive motion trajectory presentation, perception and recognition.* s.l. : Pattern Recognition, 2009.

33. **D, Hamilton J.** *Time series analysis.* s.l. : Princeton University Press, 1994.

34. *Using dynamic time warping to find patterns in time series.* **Berndt, D., Clifford, J.** s.l. : Workshop on Knowledge Discovery and Databases, 1994.

35. **Deller, J. R.** *Discrete-time processing of speech signals.* s.l. : Wiley-Interscience, 2000.

36. *Considerations in dynamic time warping algorithms for discrete word recognition.* **Rabiner, L., Rosenberg, A. and Levinson, S.** 6, s.l. : Acoustics, Speech and Signal Processing, IEEE Transactions on, 1978, Vol. 26.

37. **Kruskal, J., and Liberman, M.** *The symmetric time-warping problem: From continuous to discrete. In Time Warps, String Edits, and Macro Molecules: The Theory and Practice of Sequence Comparison.* s.l. : Addison-Wesley, 1983.

38. *Minimum Prediction Residual Principle Applied to Speech Recognition.* **Itakura, F.** 1, s.l. : IEEE Transactions on Acoustics,Speech and Signal Processing, 1975, Vols. ASSP-23.

39. *Derivative dynamic time warping.* **Pazzani., Eamonn Keogh and M.J.** Chicago, Illinois : In First Intl. IAM Intl. Conf. on Data Mining, 2001.

40. *Curve matching, time warping, and light fields: New a lgorithms for computing similarity between curves.* **A. Efrat, Q. Fan, and S. Venkatasubramanian.** s.l. : Journal of Mathematic Imaging and Vision, 2007.

41. *Applied smoothing techniques for data analysi.* **Bowman, A. W., and A. Azzalini.** s.l. : Oxford Science Publications, 1997.

42. *Image Quality Assessment: From Error Visibility to Quality Assesment.* **Zhou Wang, Member, IEEE, Alan Conrad Bovik, Hamid Rahim Sheikh, Eero P. Simoncelli.** 4, s.l. : IEEE Transactions on Image Processing, 2004, Vol. 13.

43. *Motion texture: a new motion based video representation.* **Ma, Yu-Fei and Zhang, Hong-Jiang.** 2002. 16th International Conference on Pattern Recognition. Vol. 2, pp. 548- 551.

44. *Efficient video similarity measurement with video signature.* **Cheung, S.-S. and Zakhor, A.** 1, s.l. : IEEE Transactions on Circuits and Systems for Video Technology, 2003, Vol. 13.

45. *Object Trajectory-Based Activity Classification and Recognition Using Hidden Markov Models.* **Bashir, F.I., Khokhar, A.A. and Schonfeld, D.** 7, 2007, IEEE Transactions on Image Processing, Vol. 16, pp. 1912-1919.

VIII Annex

VIII.1 Proposed algorithm clustering results

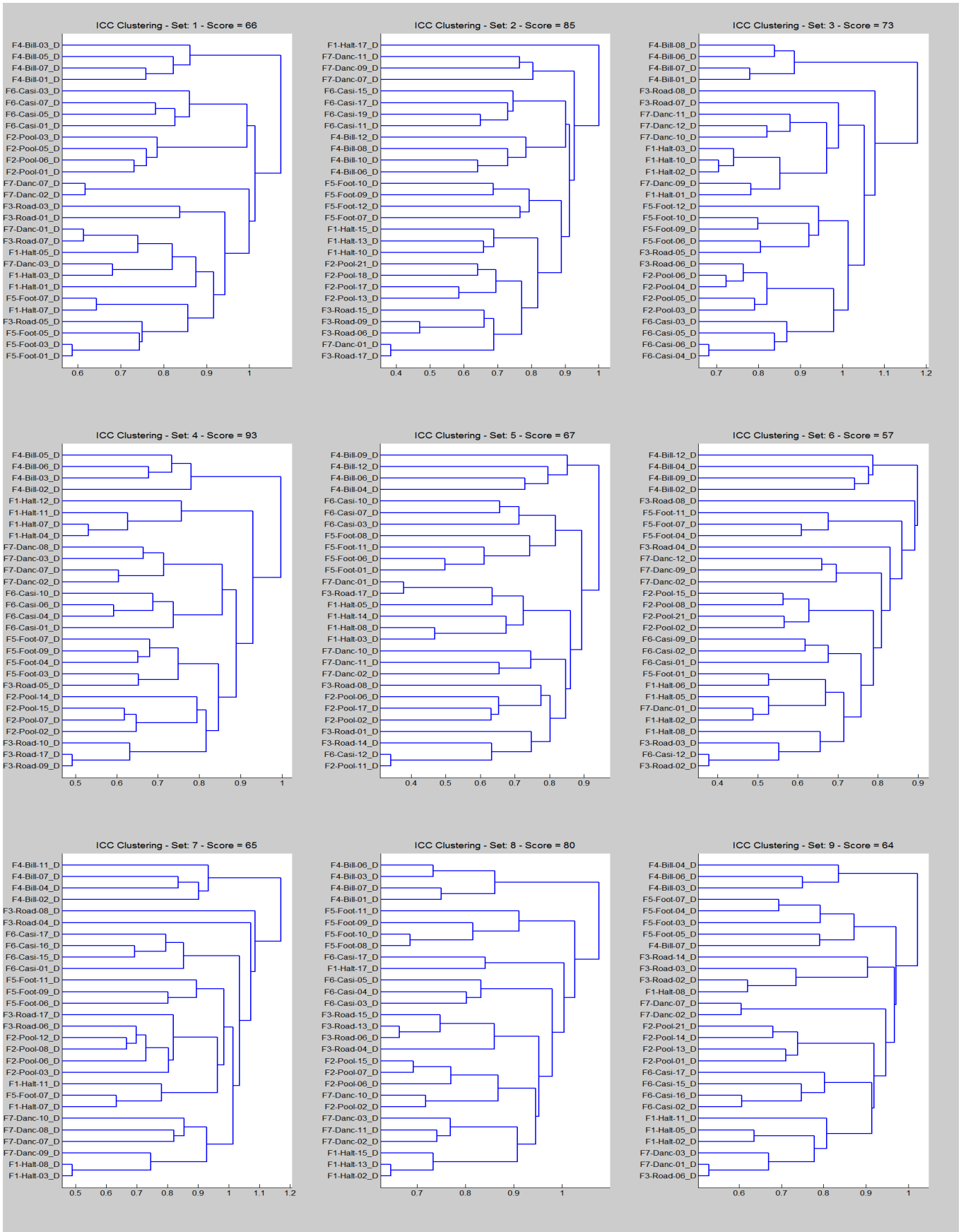


Fig. 54. Proposed algorithm clustering results

VIII.2 SSIM clustering results

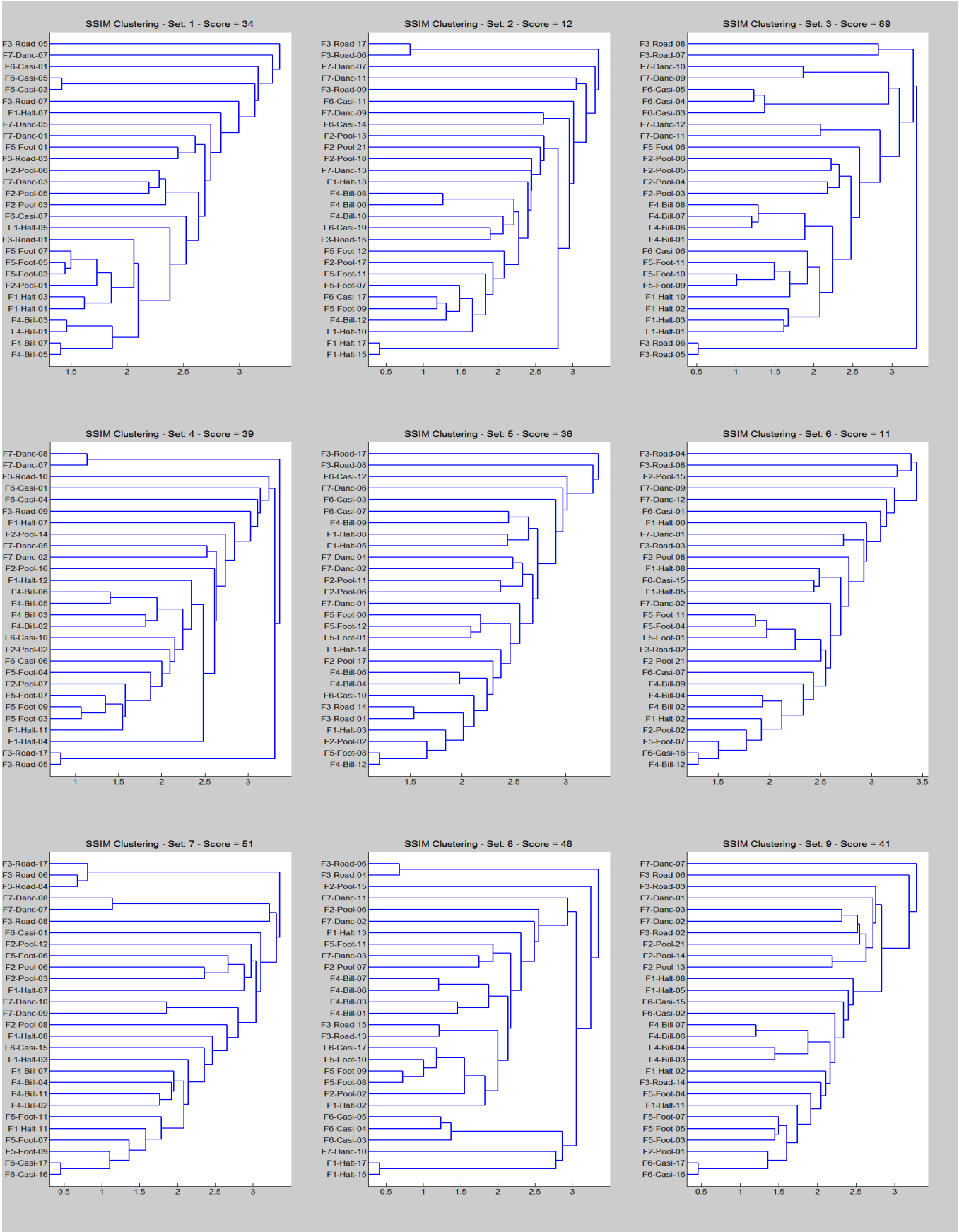


Fig. 55. SSIM clustering results

VIII.3 ICC clustering

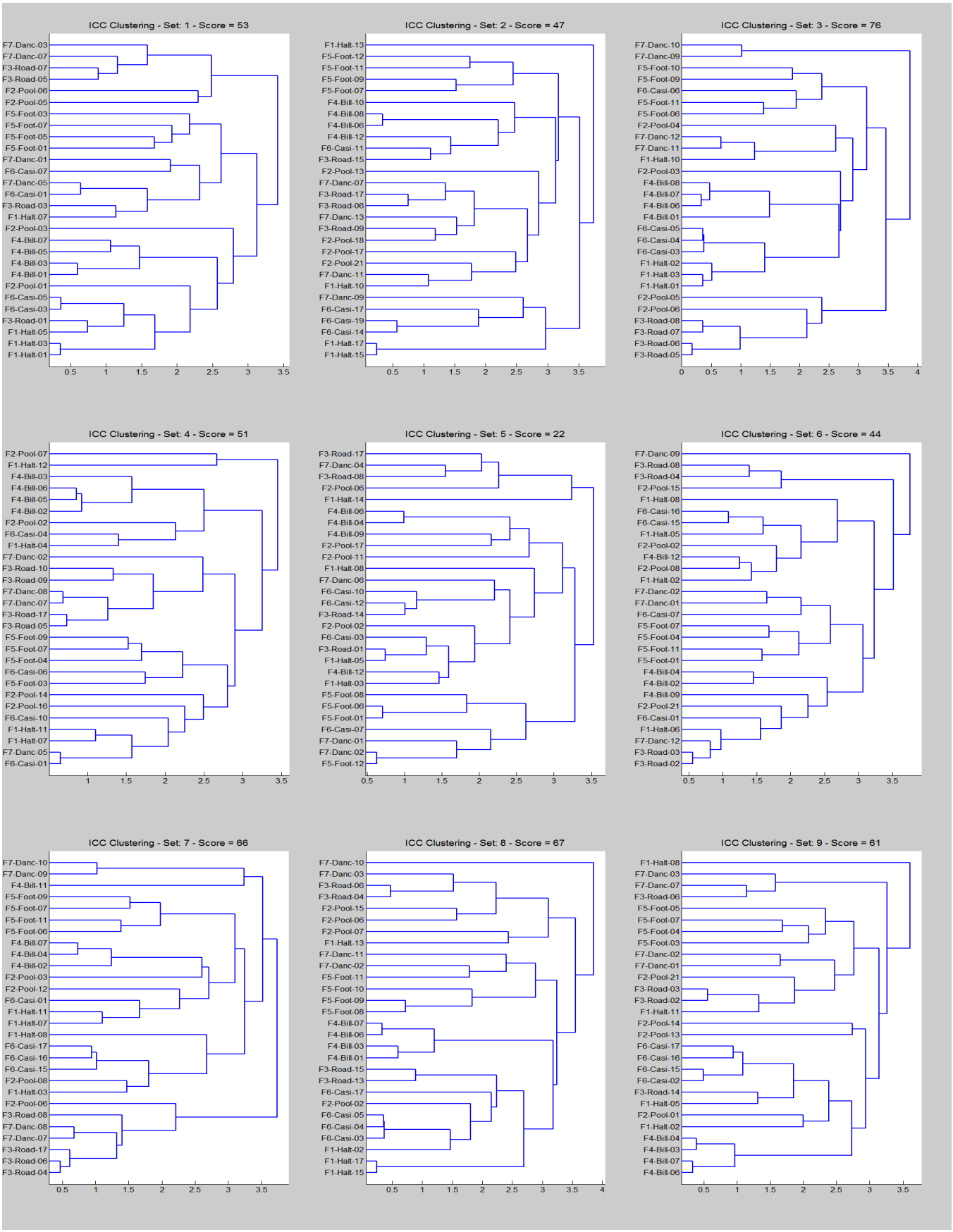


Fig. 56. ICC clustering results.

VIII.4 Motion Texture clustering results

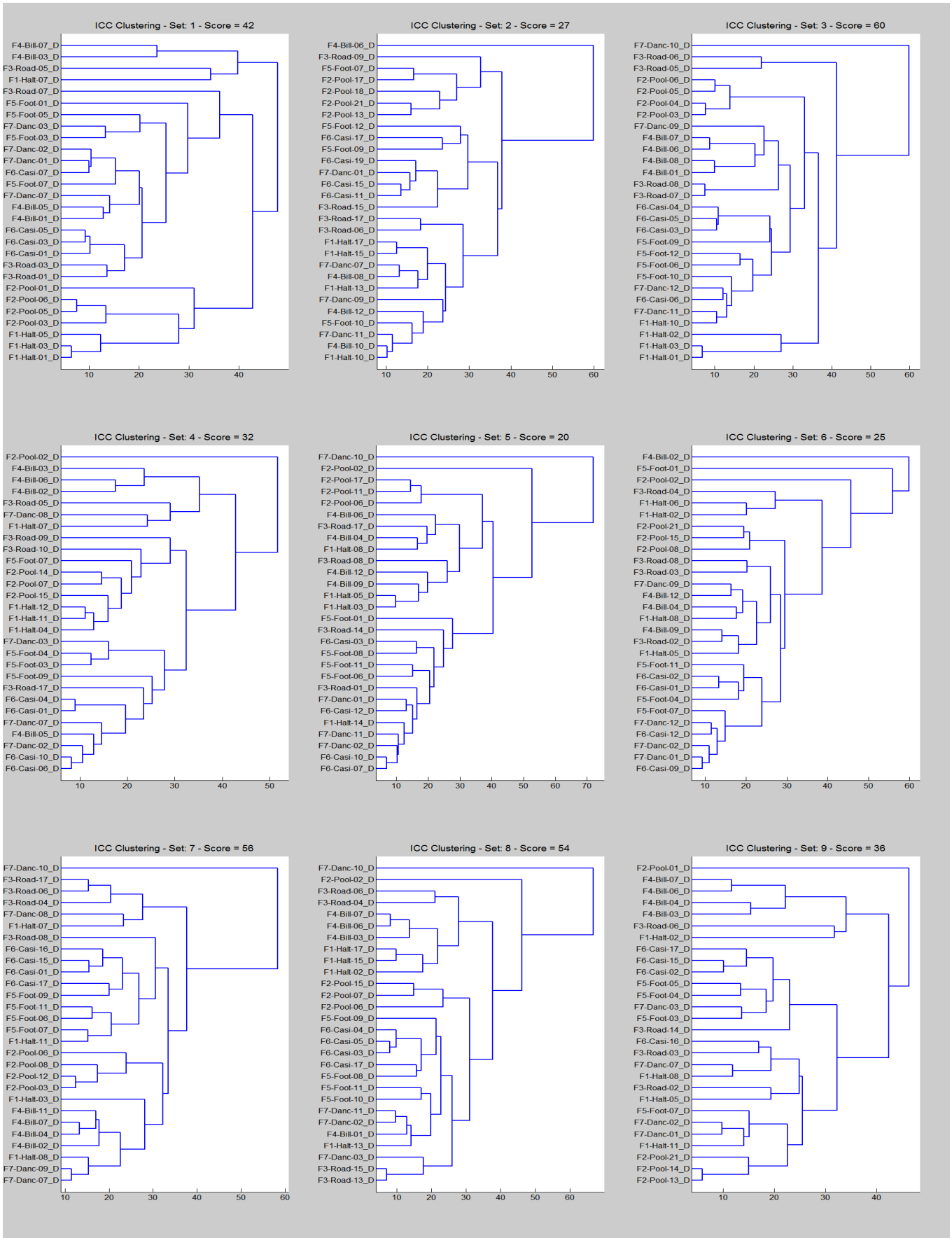


Fig. 57. Motion Texture clustering results

VIII.5 Video Database thumbnails


Video Set: 5
Video Set: 6
Video Set: 7
Video Set: 8
Family 1
Family 2
Family 3
Family 4
Family 5
Family 6
Family 7

VIII.6 Video Database Links

VIII.6.1 Video database used for clustering

The full database can be downloaded from this link: http://www.megaupload.com/?d=0WEWKSDT

Videos can be watched independently from the link below. Caution: for original CIF resolution videos, use the previous link to download the full database, the links in the table below are to 240 lines streaming youtube compressed videos.

F1: Weightlifting	
F1-Halt-01	http://www.youtube.com/watch?v=bY58XQnKhuY
F1-Halt-02	http://www.youtube.com/watch?v=MQ_miG7ryiI
F1-Halt-03	http://www.youtube.com/watch?v=jyNv_c1TKBo
F1-Halt-04	http://www.youtube.com/watch?v=AleAZllNzi4
F1-Halt-05	http://www.youtube.com/watch?v=XXYztzYMm_U
F1-Halt-06	http://www.youtube.com/watch?v=N_OUyTp7r24
F1-Halt-07	http://www.youtube.com/watch?v=1D0zMzWfdZA
F1-Halt-08	http://www.youtube.com/watch?v=rXyWvV_UXsE
F1-Halt-09	http://www.youtube.com/watch?v=xH8fIQgmgw0
F1-Halt-10	http://www.youtube.com/watch?v=TBTa6Smdq3I

F1-Halt-11	http://www.youtube.com/watch?v=pgFtELYDUQM
F1-Halt-12	http://www.youtube.com/watch?v=yQfQ5NSYutE
F1-Halt-13	http://www.youtube.com/watch?v=RY49BDoJJlY
F1-Halt-14	ww.youtube.com/watch?v=oAt25DnNcAI
F1-Halt-15	http://www.youtube.com/watch?v=vRqMTo7976U
F1-Halt-16	http://www.youtube.com/watch?v=9k4Ckf_svWs
F1-Halt-17	http://www.youtube.com/watch?v=SdxjMzDuXys
F2: Pool Diving	
F2-Pool-01	http://www.youtube.com/watch?v=89NiQd4anAQ
F2-Pool-02	http://www.youtube.com/watch?v=_FoUl_vYIQE
F2-Pool-03	http://www.youtube.com/watch?v=lYeVnJ6qfgE
F2-Pool-04	http://www.youtube.com/watch?v=F0CJ4CqO6UQ
F2-Pool-05	http://www.youtube.com/watch?v=AfKBeg6dM9E
F2-Pool-06	http://www.youtube.com/watch?v=EGmk5hS6OwM
F2-Pool-07	http://www.youtube.com/watch?v=BsgQ3NnFpWQ
F2-Pool-08	http://www.youtube.com/watch?v=utR0CSqpCac
F2-Pool-09	http://www.youtube.com/watch?v=JvCpH5eXU94
F2-Pool-10	http://www.youtube.com/watch?v=IjRPTRtCBPg
F2-Pool-11	http://www.youtube.com/watch?v=eBu9Y9MRLUg
F2-Pool-12	http://www.youtube.com/watch?v=w9UL2ccliuQ
F2-Pool-13	http://www.youtube.com/watch?v=i73_UXM9WBg
F2-Pool-14	http://www.youtube.com/watch?v=RE21kgSOivs
F2-Pool-15	http://www.youtube.com/watch?v=Jeu28Jbh_JI
F2-Pool-16	http://www.youtube.com/watch?v=MWZlsGkJQUc
F2-Pool-17	http://www.youtube.com/watch?v=NuNk5VhzSFE
F2-Pool-18	http://www.youtube.com/watch?v=8aDG19XnHMo
F2-Pool-19	http://www.youtube.com/watch?v=ZC-SASJkcFI
F2-Pool-20	http://www.youtube.com/watch?v=1Jpgx5TBFGk
F2-Pool-21	http://www.youtube.com/watch?v=CpxFgjFP5o0
F3: Road Surveilance	
F3-Road-01	http://www.youtube.com/watch?v=uI0js4dZ7NY
F3-Road-02	http://www.youtube.com/watch?v=r7ILd9512AA
F3-Road-03	http://www.youtube.com/watch?v=zsbRaL0eHCE
F3-Road-04	http://www.youtube.com/watch?v=Q35yHyAnOh4
F3-Road-05	http://www.youtube.com/watch?v=LVgFPRghbp8
F3-Road-06	http://www.youtube.com/watch?v=99ilke_gTvk
F3-Road-07	http://www.youtube.com/watch?v=5IcLH1LjDws
F3-Road-08	http://www.youtube.com/watch?v=3wAT0Y-Y0ts
F3-Road-09	http://www.youtube.com/watch?v=vuEY6xMayTI
F3-Road-10	http://www.youtube.com/watch?v=0pa_-E5_xz8
F3-Road-11	http://www.youtube.com/watch?v=Fa3gMR8OZt0
F3-Road-12	http://www.youtube.com/watch?v=Y8kVgGh6oxU
F3-Road-13	http://www.youtube.com/watch?v=nwLO5m2WUnE

F3-Road-14	http://www.youtube.com/watch?v=OLXiyR688zc
F3-Road-15	http://www.youtube.com/watch?v=5YSV8R58aSk
F3-Road-16	http://www.youtube.com/watch?v=yurvpnCzsXg
F3-Road-17	http://www.youtube.com/watch?v=sGNwG_YhjLA
F4: Billard	
F4-Bill-01	http://www.youtube.com/watch?v=pC2WOACwFuw
F4-Bill-02	http://www.youtube.com/watch?v=x7KFCKIB0Uw
F4-Bill-03	http://www.youtube.com/watch?v=GAWGNfs371Q
F4-Bill-04	http://www.youtube.com/watch?v=ZjvgVpitZyw
F4-Bill-05	http://www.youtube.com/watch?v=PcECNKdPlus
F4-Bill-06	http://www.youtube.com/watch?v=kYb_6o_gRaI
F4-Bill-07	http://www.youtube.com/watch?v=Uf1w5EBI8tM
F4-Bill-08	http://www.youtube.com/watch?v=rKDafTgoZYw
F4-Bill-09	http://www.youtube.com/watch?v=DVfZwAfto1s
F4-Bill-10	http://www.youtube.com/watch?v=Al8XsJcn7Vg
F4-Bill-11	http://www.youtube.com/watch?v=oy9gGbbxtis
F4-Bill-12	http://www.youtube.com/watch?v=vX6duThnVDU
F5: Soccer	
F5-Foot-01	http://www.youtube.com/watch?v=3GRIDBCV1KY
F5-Foot-02	http://www.youtube.com/watch?v=jWYqEs8fSB4
F5-Foot-03	http://www.youtube.com/watch?v=O6twb9NuHqU
F5-Foot-04	http://www.youtube.com/watch?v=csKLjHATSk0
F5-Foot-05	http://www.youtube.com/watch?v=wHiBhOsF7lk
F5-Foot-06	http://www.youtube.com/watch?v=e0ikZo0M7M0
F5-Foot-07	http://www.youtube.com/watch?v=UyLLmCv6rxM
F5-Foot-08	http://www.youtube.com/watch?v=AgCT5novZ64
F5-Foot-09	http://www.youtube.com/watch?v=FwRrlfqRMYQ
F5-Foot-10	http://www.youtube.com/watch?v=D-Y8bf2RM2E
F5-Foot-11	http://www.youtube.com/watch?v=Y0uWpYBwLb0
F5-Foot-12	http://www.youtube.com/watch?v=BE5BUwjnUTE
F6: Casino	
F6-Casi-01	http://www.youtube.com/watch?v=MZwq5yGmOPw
F6-Casi-02	http://www.youtube.com/watch?v=CLNTf4n-QNk
F6-Casi-03	http://www.youtube.com/watch?v=vLplPmPBX5Y
F6-Casi-04	http://www.youtube.com/watch?v=WPV9bA4Klyg
F6-Casi-05	http://www.youtube.com/watch?v=UrzC0LeDBOY
F6-Casi-06	http://www.youtube.com/watch?v=DXwOBKBIonQ
F6-Casi-07	http://www.youtube.com/watch?v=0DHyWUBs88o
F6-Casi-08	http://www.youtube.com/watch?v=gqLotH3T46o
F6-Casi-09	http://www.youtube.com/watch?v=1LCjyyR5OBE
F6-Casi-10	http://www.youtube.com/watch?v=e4GP1tbKe48
F6-Casi-11	http://www.youtube.com/watch?v=2rlprfGlvKg

F6-Casi-12	http://www.youtube.com/watch?v=tN4r2bkI70M
F6-Casi-13	http://www.youtube.com/watch?v=qyGno-ZUg5k
F6-Casi-14	http://www.youtube.com/watch?v=YHl1NUfuVdM
F6-Casi-15	http://www.youtube.com/watch?v=OXdp-Hl6IjA
F6-Casi-16	http://www.youtube.com/watch?v=bibR931ZAP8
F6-Casi-17	http://www.youtube.com/watch?v=UFjkPWq558U
F6-Casi-18	http://www.youtube.com/watch?v=KWtVzdDKmxc
F6-Casi-19	http://www.youtube.com/watch?v=14HRQvD8Dq4
F7: Group Dancing	
F7-Danc-01	http://www.youtube.com/watch?v=Ab6WFcIfEko
F7-Danc-02	http://www.youtube.com/watch?v=Js1LbxehqQQ
F7-Danc-03	http://www.youtube.com/watch?v=dsmDxyyz1ss
F7-Danc-04	http://www.youtube.com/watch?v=NC9UO-cKQIQ
F7-Danc-05	http://www.youtube.com/watch?v=AXLQGQXOmrg
F7-Danc-06	http://www.youtube.com/watch?v=pzo8aqon9FM
F7-Danc-07	http://www.youtube.com/watch?v=psaY3khb4e4
F7-Danc-08	http://www.youtube.com/watch?v=Fc0BSfz_sD0
F7-Danc-09	http://www.youtube.com/watch?v=47s7UxyFpBE
F7-Danc-10	http://www.youtube.com/watch?v=KzbOSl6I42k
F7-Danc-11	http://www.youtube.com/watch?v=VdD_Th1dIYA
F7-Danc-12	http://www.youtube.com/watch?v=j5Cw0HAV7T4

Links to videos used as examples for motion extraction can be watched here:

VIII.6.2 Original sequences used in this report

Bouncing sequence: http://www.youtube.com/watch?v=XuIdyJ-lwvk

Bowing sequence: http://www.youtube.com/watch?v=7H2jJh3L2TQ

Corridor sequence: http://www.youtube.com/watch?v=659Z6URe89o

Synthetic sequence: http://www.youtube.com/watch?v=yWQa4kOACHU

Zooming sequence: http://www.youtube.com/watch?v=PFqMKppmjHs

VIII.6.3 Block matching demo sequences

Bouncing sequence: http://www.youtube.com/watch?v=BDtvjKMl30w&hd=1

Bowing sequence: http://www.youtube.com/watch?v=MTFzznP6IjQ&hd=1

Corridor sequence: http://www.youtube.com/watch?v=vc4eAdYTqjo&hd=1

Synthetic sequence: http://www.youtube.com/watch?v=J43q19pQxHo&hd=1

Zooming sequence: http://www.youtube.com/watch?v=C2TmEXY6X8M&hd=1

VIII.6.4 Workflow summary

Bouncing sequence: http://www.youtube.com/watch?v=J_M5wuSd8YQ&hd=1

Bowing sequence: http://www.youtube.com/watch?v=NLS3CdkOfw0&hd=1

Corridor sequence: http://www.youtube.com/watch?v=rSGaFHOlAqk&hd=1

Synthetic sequence: http://www.youtube.com/watch?v=-u3CmAZO2to&hd=1

VIII.7 Figures list