Multimed Tools Appl
https://doi.org/10.1007/s11042-018-5965-x

Mossar: motion segmentation by using splitting and remerging strategies

Pujana Paliyawan[1] · **Worawat Choensawat**[2] · **Ruck Thawonmas**[1]

Received: 9 July 2017 / Revised: 5 March 2018 / Accepted: 3 April 2018

Abstract This paper presents a novel approach for motion segmentation by using strategies of splitting and remerging. The presented approach, Mossar, hybridizes two existing ones to obtain their potential advantages while covering weaknesses: (1) *velocity-based*, one of the most widely used approaches that has fairly low accuracy but provides computational simplicity and (2) *graph-based*, a state-of-the-art approach that provides outstanding accuracy, yet bears high computational complexity and a burden in setting of thresholds. An initial set of key frames is generated by a velocity-based splitting process and then fed into a graph-based remerging process for refinement. We present mechanisms that improve key-frames capturing in the velocity-based approach as well as details on how the graph-based approach is modified and later applied to remerging. The proposed approach also allows users to interactively add or reduce the number of key frames to control segmentation hierarchy without the need to change threshold values and re-run segmentation, as usually done in existing approaches. Our experimental results show that the presented hybrid approach, compared to both *velocity-based* and *graph-based*, demonstrates superior performance in terms of accuracy and in comparison to *graph-based*, our approach has not only less complexity but also a lesser number of thresholds, the values of which can be much more simply determined.

Keywords Motion segmentation · Motion representation · Graph Kernel matching

✉ Pujana Paliyawan
Pujana.P@gmail.com

[1] Intelligent Computer Entertainment Lab, Graduate School of Information Science and Engineering, Ritsumeikan University, Kusatsu, Shiga-Ken, Japan

[2] Multimedia Intelligent Technology Lab, School of Information Technology and Innovation, Bangkok University, Bangkok, Thailand

Published online: 21 April 2018

1 Introduction

Temporal segmentation is a process of cutting sequential data into segments with different semantic meanings. This process serves as a crucial step to data analysis in various applications. For analysis of motion data, temporal segmentation is commonly referred to as motion segmentation (also called key-frame extraction), a process of finding key frames for separating an activity into a sequence of coherent segments, where each segment can represent a cluster of motions, a motion, or a posture, depending on expected segmentation hierarchy (Fig. 1). One must adopt a suitable motion segmentation approach, as a part of preprocessing, when building an intelligent framework for motion classification or recognition. Motion segmentation plays an important role in the performance of recognition/classification, and its sample uses are given as follows:

- To distinguish motions or a set of repetitive motions in a set of given activity data for motion understanding and analysis [26, 29].
- To detect transitions in posture[1] in a given data file and analyze movement steps for performing a motion[2] of interest. Examples are a project aiming at analyzing and archiving routines[3] of Japanese traditional dances [21] and a tool for generating Labanotation from motion data [3, 7].
- To compress motion data and summarize contents for storage and browsing [14, 15]. A search engine that recommends similar motions to a query sample can also be included here (e.g., [15]).
- To synchronize multiple temporal data, such as synchronization of motion segmentation and music tempo segmentation for creating a beat synchronous dance animation [19].

We conduct a review of existing approaches for motion segmentation by emphasizing two approaches: a velocity-based approach [3, 7, 10, 18, 21, 22, 26] that monitors the magnitude of change in body movement, and a graph-based approach [12, 13] that converts motion data into adaptive graphs and measures similarity between two graphs by using kernels. The velocity-based approach is widely accepted and used due to its implementation and computational simplicity. On the other hand, the graph-based approach is a newly proposed approach that yields outstanding performance, yet bears high computational complexity and high burden, to its users, in determining the values of several employed thresholds. Through

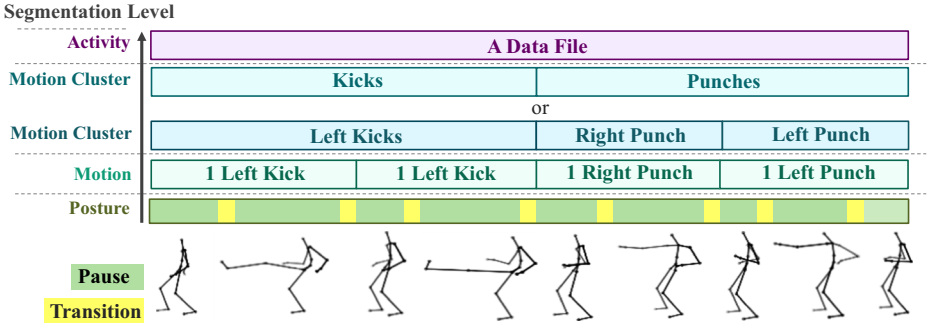

Fig. 1 Segmentation of data into different levels (three levels) of hierarchy

analysis of the two approaches, both theoretically and empirically, we come up with a hybrid approach, called Mossar (motion segmentation by using splitting and remerging strategies). Besides fast and highly accurate segmentation, the proposed approach also yields flexibility to diverse needs for segmentation and configuration simplicity—more details on these advantages are given in Section 4 and evaluated in Section 5.

The contributions of our study can be summarized as follows:

- To achieve fast and highly accurate segmentation, we focus on and cover disadvantages of the graph-based approach, a state-of-the-art approach, presented by Li and Leung [12, 13] (discussed in 2.2.4). We drastically reduce the computational complexity of this high-performance approach by modifying it from an approach for splitting to an approach for remerging, and then synergizing it with the velocity-based approach, one of the fastest segmentation approach.
- We improve the conventional velocity-based approach by adding two major mechanisms (discussed in 2.1.4), one for capturing all key frames located above the threshold line, and the other for preventing over-segmentation by filtering out insignificant key frames. As a result, the approach can better handle motions performed in a highly continuous manner.
- We demonstrate how a good combination of approaches could simplify thresholds tuning. Each of the two approaches alone requires well setting of thresholds to work effectively. On the other hand, when they are combined, their strong points support each other and relax the importance of threshold setting. The velocity-based part can extract key frames without the need to optimize its threshold for preventing over-segmentation. The remerging part will eliminate over-segmented segments, and, since it does not search for split points on every frame as done in its original version, but analyzes consecutive segments that are well-segmented enough, the graph-based approach becomes less sensitive to threshold setting.

The rest of the paper is organized as follows. In Section 2, we provide our reviews on existing approaches for motion segmentation; assumptions and limitations underlying them are also examined to explore ways for improvement or modification. Besides the velocity-based and the graph-based approaches, we compare our approach to a recently introduced self-similarity-based approach [11, 25]. In Section 3, the proposed approach, Mossar, is introduced; theory and logic on how it overcomes the aforementioned limitations in the existing approaches are also discussed. In Section 4, we first describe a technique for obtaining reliable ground truths in our experiments and then show the experimental results where Mossar is compared with existing methods. In Section 5, our conclusions, discussions, and guides for future work are given.

2 Existing approaches

2.1 Velocity-based approach

The velocity-based approach (e.g., [3, 7, 10, 18, 21, 22, 26]) uses straightforward logic and simple implementation. Pause in body movement is considered as the main segmentation feature [10], and the underlying assumption is that one typically pauses briefly while transitioning between different phases of motion [3, 7, 26]. In general, segmentation is based on the velocity curve

representing a magnitude of body movement over time. A key frame is extracted each time when the curve crosses a specified threshold—this technique is sometimes called "the zero-velocity-crossing detection [22, 26]." The velocity-based approach provides acceptable performance in a computationally efficient manner. It has been adapted to a variety of applications so far and forms the basis of the other pause-based segmentation approaches, including those for other kinds of data, such as silence or pause detection-based speech segmentation [10].

2.1.1 Computation

Velocity can be represented in several manners; for example, it can be positional velocity (e.g., [10, 18, 21, 26]) computed from relative change in a body-joint's position or angular velocity (e.g., [22]) computed from the derivative of a body-joint's angle. In many studies, multiple sets of key frames are extracted where each set is derived by using the velocity of a single body joint (e.g., [3, 7, 21, 22]) associated with the set. However, for motion segmentation considering the movement of the whole body, the total velocity can be computed as the summation of each joint's velocity (e.g., [18]), or alternatively, the summation of each selected parameter's velocity [26].

Ideally, all velocities should reach zero at the same time when the body pauses; however, because of noise, the criterion for zero-velocity-crossing detection is relaxed [26]. In practice, minimum-speed-crossing detection [21] is employed where a key frame is extracted each time when the velocity curve crosses a significant threshold (denoted as *sigThre*). This threshold forms a so-called threshold line. The period in which the magnitude of the speed of a joint is lower than the threshold line is taken as a resting period [3, 7] (resting state [26], zero-velocity period [18], or transition segment [11]). On the other hand, the period in which the magnitude of the speed of a joint is higher than the threshold line is called a moving period (or action segment [11]).

Regardless of variety in implementation, we conclude that most velocity-based methods share three common characteristics, which form the conventional velocity-based approach. First, segmentation is based on a velocity curve. Second, *sigThre*-crossing is commonly used as a technique for extracting key frames. Third, a resulting segment can be classified as either a resting-period segment or a moving-period segment.

2.1.2 Computational complexity

As a data file is only scanned from the first to the last frame, the big-O of the whole method is O(F), where F is the number of frames in the input data file.

2.1.3 Configuration simplicity

The threshold, *sigThre*, must be properly determined; a too high value of this threshold would make the method incapable of capturing key frames at all when the pausing time is short or unclear while a too low value usually leads to over segmentation. It was also reported that the approach faced problems in segmentation of very slow motions [3].

2.1.4 Our improvements

In our approach, for the splitting part, we improve the conventional velocity-based approach by adding two major mechanisms, one for capturing all key frames located above the threshold line, and the other for preventing over-segmentation by filtering out insignificant key frames.

The first mechanism monitors all local minima and maxima on the velocity curve; this mechanism is inspired by existing works [1, 18, 21, 26] in which a curve used in computation may represent either the sum of squared velocity [26], a joint-positional velocity [21], a joint angle [1], or the total velocity [18]. The second mechanism helps avoiding over-segmentation by creating a bounding envelope for accepting only key frames whose change from their preceding key frame is significant.

Before adding the said mechanisms, we also apply normalization for making the threshold become scalable to various velocities across a given set of data files.

2.2 Graph-based approach

A graph-based approach for motion segmentation has been recently introduced by Li et al. [12, 13]. This approach recursively performs motion segmentation. Each recursion step aims to split a segment of interest into two segments (Fig. 2) through iteration of graph construction and comparison for finding the optimal split point. Candidate split points are tested, and in each test, the similarity between two consecutive segments, the one on the left and the one on the right of the split point, is measured. The selected candidate is the one that minimizes the similarity between the two consecutive segments while satisfying given conditions—such as, the similarity between the split segments must be less than a specified threshold, or the number of split points has not reached a desired number. Each segment is represented by a graph, and therefore the similarity between any two consecutive segments is measured by comparing the two graphs of interest by using graph kernels. The approach provides superior segmentation accuracy over existing approaches; however, it has high computational complexity in terms of time and memory usage and requires setting of several thresholds.

We summarize the graph-based approach by Li et al. in four subsections. Section 2.2.1 describes computation in the graph-based approach, focusing on (1) graph construction and similarity measurement, and (2) segmentation processes based on a recursive and iterative concept for finding split points. Sections 2.2.2 and 2.2.4 describe two important issues that our hybrid approach, Mossar, is designed to overcome: computational complexity and setting of thresholds. Section 2.2.4 describes our modifications to the graph-based approach when it is used as a remerging process in Mossar.

2.2.1 Computation

To understand the graph-based approach clearly, we review it through two separated parts (two groups of main concepts). The first part is about graph construction by using motion data and

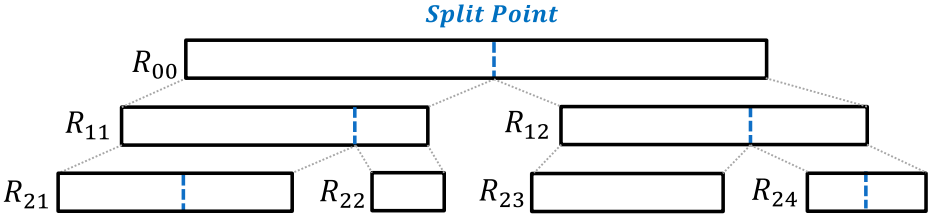

Fig. 2 Recursive segmentation, in which each recursion is identified by two indices: the first is the level of recursion, and the second is the index within the level. The top level covers the entire input data span and is denoted by R_{00}

about comparison of graphs to measure their similarity. The second part is about segmentation, in which a recursive and iterative concept is used for finding split points.

Graph construction and similarity measurement A segment consists of raw data that represent a sequence of body joints' positions in three-dimensional coordinates. In the graph-based approach, data in a segment are summarized in the form of graph in the following. A vertex represents a joint and is associated with Temporal Pyramid Covariance (TPC) [12], a vector of 6 s dimensions where s is the number of segments in the segmentation pyramid (cf. Fig. 3) and 6 is number of elements in the upper triangular of the 3×3 covariance matrix of the coordinates of the joint over a segment of interest of length T. An edge is associated with Range of Joint Relative Distance (RRJRD), a scalar value representing the range of the distance between the two corresponding joints over T. In Li et al., the graph of a given segment is represented by the top N edges in terms of RRJRD. Figure 3 shows an example where $N = 4$ and $s = 3$. The similarity between a pair of two graphs, or two motion segments, is calculated using a graph kernel (see (11) in [12]) considering their differences in terms of both TPC and RRJRD in all possible common paths of both graphs.

Segmentation Each recursion searches for a splitting point through a number of iterations, in each of which two graphs, one for the preceding segment and the other for the proceeding one, are constructed and compared to obtain the similarity indicating the fitness of that point. Fig. 4a provides an example where two consecutive segments are initialized at the first iteration to each have the pre-determined minimum size, C_{min}, of 20 frames; they are then compared, and the *Similarity* (i.e., the value returned by a graph kernel in use) of 1.3 is obtained. For subsequent iterations, the size of the first segment is expanded incrementally according to a given search precision P while the second segment is shifted rightward accordingly. A split point is found where the *Similarity* is minimized (at frame #80 in this case). The above process recursively continues by finding more split points in the already-split segments (see Fig. 4b). A split point will not be accepted if its *Similarity* is greater than a specified threshold, and only a segment whose size is greater than $2C_{min}$ can be further split—for example, recursions R_{21} and R_{22} in Fig. 4b abort because the least *Similarity* found in R_{21} is greater than the threshold and because R_{22} is smaller than the aforementioned $2C_{min}$.

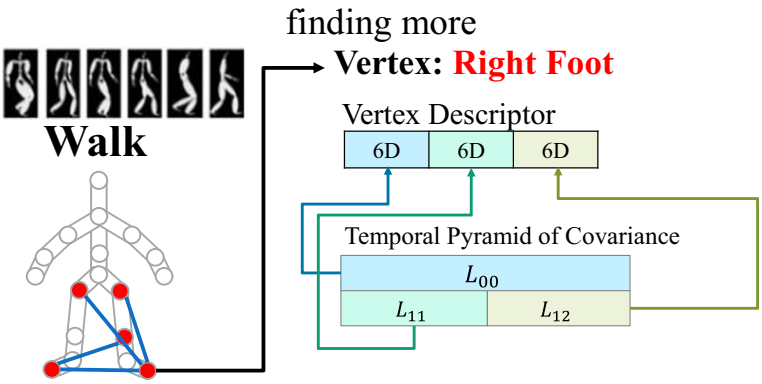

Fig. 3 Derivation of a vertex descriptor for body joint "Right Foot" in a graph representing a segment of motion "Walk"

2.2.2 Computational complexity

Big-O for the whole segmentation process is $O(N\cdot\sqrt{N}!\cdot F^2)$. The computational complexity of the graph-based approach is represented as a multiplication between computational complexity for comparing one pair of graphs and the number of times a pair of graphs are compared. They are $N\cdot\sqrt{N}!$ and F^2 in the big-O respectively. Details on calculation are given below.

- ***Computational complexity for comparing one pair of graphs*:** This is computational of a graph kernel, and is mainly affected by the value of threshold N. We consider the number of walk-kernel calls as a unit for measuring this complexity. Walk kernel is a kernel for measuring the similarity between two matched walks from two different graphs; in both Li et al. and our studies, all input fed to this kernel are paths, or in other words, walks in each of which all vertices are distinct. A graph is constructed based on a given number of edges (denoted as N). In the worst case, all paths in the two graphs are matched, and the number of walk-kernel calls is equal to the maximum possible number of paths in a graph (denoted as P)—we need to compute P based on the given N. A simple trick is needed because while a graph is constructed based on the value of N, knowing only N does not provide enough information to derive such P. Our solution is to assume that a graph of interest as a complete graph and then demonstrate growth in P over N.
  - ***Computational complexity for a complete graph of N edges*:** Given N, the number of vertices, denoted as V, in a complete graph is $\frac{1+\sqrt{(1+8N)}}{2}$ for any N resulting in a positive integer of V. The total number of paths having the length equal to or larger than 1 between any two vertices in a complete graph of V nodes is $[e(V-2)!]$ for $V\geq 3$ [23], where e is Napier's constant and $[x]$ denotes the largest integer $\leq x$, and there are $(\frac{V(V-1)}{2})$ such pairs of vertices. As a result, the number of paths, denoted as P, that must be examined in a complete graph of V nodes is $\left(\frac{V(V-1)}{2}\right)[e(V-2)!]+V$, where the added V paths are those with the length of zero (empty paths), leading to the big-O of $O(N\cdot\sqrt{N}!)$.
- ***The number of times a pair of graphs are compared*:** In a recursion, the number of graph-kernel calls is $\left\lfloor\frac{f-2C_{min}}{P}+1\right\rfloor$, where f is the number of frames in the recursion and $\lfloor x\rfloor$ is the floor operator; for example, in Fig. 4b, R_{00} and R_{11} has f of 150 and 80, leading to 12 and 5 graph-kernel calls, respectively, for $C_{min}=20$ and $P=10$. The whole segmentation

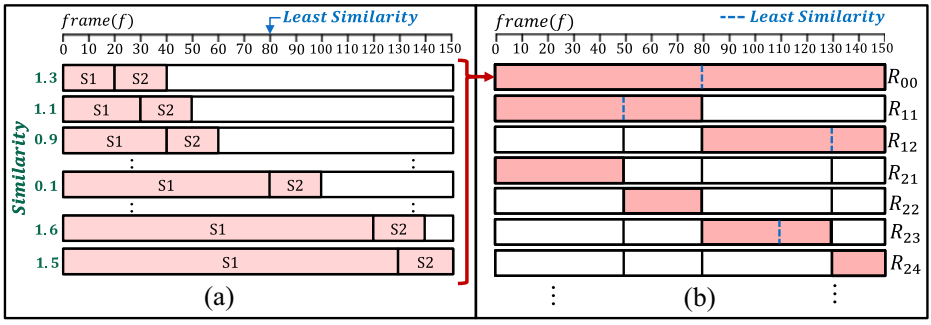

Fig. 4 An example of segmentation; **a** the first recursion by which a split point is found at f_{80} ($C_{min}=20$, $P=10$), and **b** results after the first seven recursions where shaded regions represent segments of interest and vertical dot lines in those regions indicate split points (i.e., extracted key frames)

process has the possible maximum number of recursions of $\left\lfloor \frac{F}{C_{min}} - 1 \right\rfloor$, where F is the number of frames in the entire input data span ($F \geq f$). As a results, we conclude that the number of times a pair of graphs are compared has computational complexity of $O(F^2)$.

2.2.3 Configuration simplicity

There are five important thresholds whose values must be determined: N (the number of edges for graph construction), L (the number of layers for constructing a temporal pyramid of covariance), C_{min} and C_{max} (the allowed minimum and maximum size of segment), and *Min_threshold* (the similarity threshold used to filter out insignificant key frames). According to Li and Leung [12], N and L are the two most important thresholds; an increase in N significantly boosted the segmentation accuracy until it reached the value of about 45, while an increase in L slightly improved the accuracy. The rest were not examined; they were specifically predefined for each particular dataset. We, however, found that optimization of *Min_threshold* should also take the value of N into consideration because *Similarity* is computed as the summation of different p-th order graph kernels (see (11) in [12])—a similarity of two sets of paths with the path length of p is measured by a p-th order graph kernel, after which similarities from different path lengths are summed up. Because increase in N will increase the longest possible path length and hence affect the value of *Similarity*, optimization of *Min_threshold* is not that straightforward.

2.2.4 Our improvements

Summarized below are our modifications to the graph-based approach in order to use it as the remerging part of Mossar:

(1) reducing the computational complexity by eliminating unnecessary kernel calls based on our assumption that a key frame exists where the velocity changes, meaning that all key frames during movement with stable velocity do not need to be considered;
(2) making the size of the second segment (the right segment or S2 in Fig. 2a) more flexible—as the original approach fixes it at C_{min}, regardless of the fact that the segment length can vary across different motions, and even across the same motion; and
(3) reducing the number of thresholds in use, especially, by eliminating C_{min}, C_{max}, and *Min_threshold*, the robustness of all of which was not tested by Li and Leung [12].

The above modifications are possible due to our use of the graph-based approach for the remerging process rather than the splitting one. Namely, only the key frames and segments derived by the splitting process are to be processed. Thereby, not every single point needs to be tested, and the segment size is flexible. In addition, thresholds C_{min} and C_{max} become unnecessary, and *Min_threshold* can be omitted if the hierarchy of segmentation is controlled interactively (further discussed in 3.1.2.2).

2.3 Self-similarity-based approach

Another approach is based on analysis of self-similar structures, introduced by, for example, Vögele et al. [25] and recently improved by Krüger et al. [11]. In this approach, neighborhood

graphs are applied to self-similarity-based segmentation, boiling down the problem to an efficiently solvable graph problem. We give comparisons and explanations why the work by Krüger et al. is not directly comparable to ours as follows:

(1) Their method aims at segmenting a data file into the lowest level of hierarchy without user interactions while Mossar allows its user to interactively control segmentation hierarchy without the need to change threshold values and re-run segmentation.
(2) They used an assumption that human activities are of repetitive nature—it was clearly stated by them that the limits of finding postures are reached when input sequences contain only non-repetitive activities. On the contrary, our approach works regardless of the number of times motions are repeated. Note that detection of such non-repetitive motions is crucial for real-time segmentation in cases such as motion gaming where motions are non-repetitive and unpredictable.
(3) They introduced a preprocessing technique called "feature bundling" for accommodating modalities of input while in this work we specialize our method to motion segmentation. Nevertheless, this does not imply that the usage scope of our method is limited because our splitting part can be applied to another kind of data in which the magnitude of change can be computed while although the remerging part of ours is based on 3D positions, most motion data can be converted into this form.

3 Proposed approach

Mossar hybridizes the velocity-based and the graph-based approaches as shown in Fig. 5. The approach consists of two major parts: splitting and remerging. The first part applies the velocity-based approach for splitting data into a set of smallest segments, from which an initial set of n key frames is obtained and fed as input to the second part. The second part applies the graph-based approach for iteratively remerging segments, from which, at each iteration, the least significant key frame is filtered out from the set of key frames. The least significant key frame at each iteration is found by searching for a pair of two consecutive segments that are most similar in terms of graph representation, as discussed in 2.2.1.1; those two segments are then remerged by removing the key frame that previously split them, and a set of key frames with decremented number of elements is obtained. As a result, if the segmentation process runs until all segments are remerged back to a single segment without additional termination criteria, such as "reaching the number of key frames specified by the user" (later discussed in 3.1.2.2), in total n sets of key frames will be obtained. From the n sets, the user can select the ones considered most suitable to his or her desired segmentation levels or usages (see [8] for a demo video on how to interactively control segmentation hierarchy.

Mossar is implemented in the following. The splitting part is derived and improved from the conventional velocity-based approach mentioned in 2.1.1 while the remerging part is implemented based on a graph-based approach by Li and Leung [12]. Here are potential advantages of our method:

- ***Fast and highly accurate segmentation***: Mossar is aimed for high segmentation accuracy as the graph-based method but with low computational complexity as the velocity-based method.
- ***Flexibility to diverse needs in segmentation***: Different applications require different levels of segmentation hierarchies [15, 16]; for example, from a given activity as shown in Fig. 1,

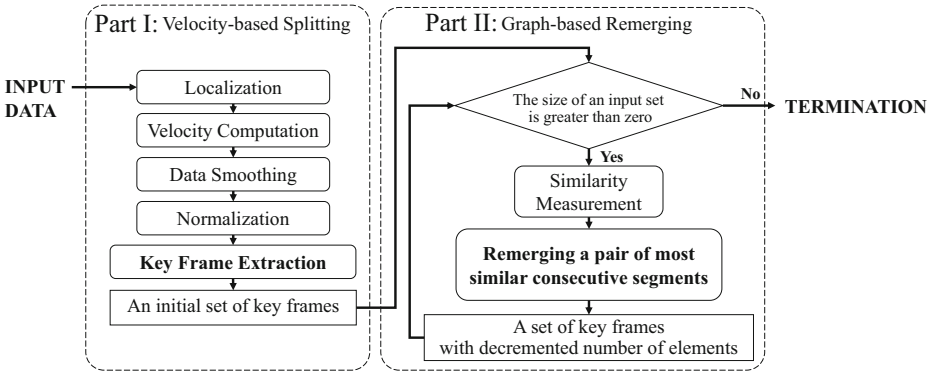

Fig. 5 The processing steps of Mossar where rectangles represent outputs of the segmentation process. Steps with bold text are major steps in which our improvements are introduced

some applications monitor every change in posture by, say, counting every foot step, punch, and kick, and, thereby, transitioning time must be distinguished from pausing time. On the other hand, some applications may only need the information on motion clusters. Mossar satisfies such diverse needs by offering results with an adjustable number of key frames; from the resulting n sets of key frames, users can select the set(s) they want for their application without the necessity to adjust any threshold values and re-run segmentation processes, as done in existing approaches.

- ***Configuration simplicity***: Our design of Mossar focuses on minimizing the number of thresholds in use and user burden to specify their values.

3.1 Design and implementation

In this section, we describe the processing steps of Mossar (Fig. 5) in two subsections: velocity-based splitting and graph-based remerging. In each subsection, design and implementation of the core components are described. It is noted that the velocity-based splitting part presented in this work can also be used as a stand-alone approach for segmentation, because it can be considered as an improved velocity-based approach, and is referred to as $\text{Mossar}_{\text{Splitting}}$.

3.1.1 Velocity-based splitting ($\text{Mossar}_{splitting}$)?

Preprocessing

(1) ***Localization***: $\text{Mossar}_{\text{Splitting}}$ takes a sequence of all considered body joints' positions as input. These positions are affected by the standing location of the subject. Similarly to some existing studies [3, 21], this step localizes the coordinates by subtracting them by the position of the body center to make the body joints invariant to any standing locations.

(2) ***Velocity Computation***: The velocity of a body joint at time t is defined as the summation over a specific time span of the Euclidian distances each between a pair of consecutive

frames (1), where *fps* is the frame rate and (x_i, y_i, z_i) are the 3D coordinates of the body joint at time *i*. The velocity of each joint is then summed up for the total velocity, which will be used in all subsequent processes.

For example, a dataset from the CMU database used in this study has *fps* of 120, which means that each second of human movement is described by 120 frames of skeleton data. The velocity of a body joint at a time of interest *t* is represented as an accumulated amount of displacement over 120 frames, counting from the $(t-119)^{th}$ frame to the frame at time *t*.

$$v_t = \sum_{i=t-fps+1}^{t} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2} \quad (1)$$

(3) ***Data Smoothing***: Data smoothing is commonly adopted for smoothing out short-term fluctuations (e.g., [18, 21, 26]), and hence reducing effects from noise. One possible technique is a moving average, whose size of sliding window is typically determined based on the frame rate of motion data.

(4) ***Normalization***: Typically, the significant threshold (*sigThre*) is defined in a unit of meters (e.g., [3, 7]), so its value must be determined according to sensor modalities and motion types in a given data file. To make the threshold scalable to various velocities across a set of data files of interest, each total velocity is normalized into a scale of [0, 1] by Min-Max Normalization, using the minimum and maximum values within the data file.

Key-frame extraction Key frames are extracted based on the velocity curve by two techniques: *sigThre*-crossing and local-minima capturing. By using *sigThre*-crossing, key frames are located at places where a so-called significant line is crossed. Figure 6a shows an example of key-frame extraction for a motion "Hadouken" where four key frames are extracted producing five segments, among which segments s_1, s_3 and s_5 are resting-period segments while s_2 and s_4 are moving-period segments. Figure 6b shows another example where the subject performs the same motion in a highly continuous manner, in which k_2 cannot be detected by the *sigThre*-crossing technique. However, by using the local-minima-capturing technique, inspired by a number of existing studies [1, 18, 21, 26], the missing key frame, k_2, can be recovered.

In brief, the *sigThre*-crossing technique, as done in the conventional velocity-based approach, is used to obtain resting-period segments and moving-period segments. Each moving-period segment is then analyzed, and sub-moving-period segments therein are extracted by the local-minima capturing technique, which is our second improvement to the conventional approach. In addition, for filtering out insignificant minima, our third improvement, during an analysis of a moving-period segment of interest, we adapt a concept of bounding envelope in the following.

Bounding envelope Bounding envelope represents the region for expected values of a line graph. It was commonly used for analysis of trajectory similarities [24] and approximation of time series [9]. In this work, in order to filter out insignificant key frames and thus prevent over-segmentation, we introduce the spatial branch and bound method [17], which discards relatively-less-significant minima, to bounding envelope. $\text{Mossar}_{\text{Splitting}}$ generates *Ceiling* and *Floor* that form a bounding envelope by using Algorithm 1 where the input V_t represents a normalized velocity at time *t*; this envelope is used for determining whether minima/maxima of interest are significant.

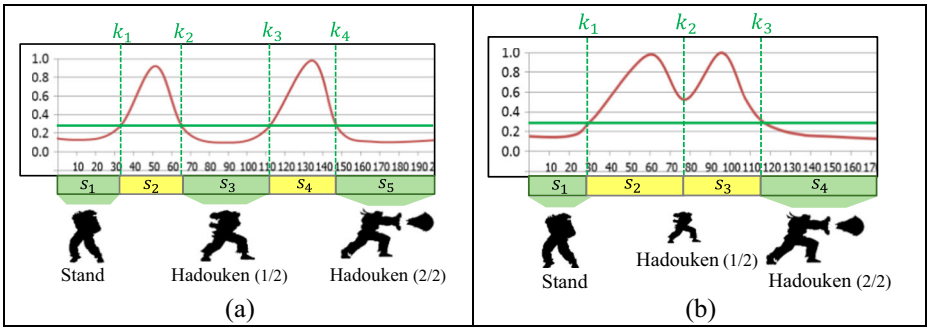

Fig. 6 Extraction of key frames for a motion "Hadouken"; **a** the motion is performed with clear pauses between stages of motion, and **b** the motion is swiftly and continuously performed where k_2 can be detected by the local-minima-capturing technique, but not *sigThre*-crossing

Analysis of a moving-period segment for extraction of sub-moving-period segments The goal here is to find significant local minima for extracting key frames during a moving-period segment using the capturing-local-minima technique. These minima will be captured and added to the list of key frames (denoted as *List*). Details on detection of significant local minima are provided below. An example in Fig. 7 is given together with Table 1 for illustration purposes.

- ***Detection of a candidate***: A candidate (denoted as *c*) represents the timing where a local minimum is found at *Floor* (e.g., at $t = 1144$ in Table 1). Such a local minimum is called significant local minimum while local minima not found at *Floor* will be ignored. A candidate will be either accepted and added to *List* or discarded.

Algorithm 1 Derivation of *Ceiling* and *Floor* at time t

```
procedure updateEnvelope()
    if V_t > Ceiling_{t-1} then
        Ceiling_t = V_t
        Floor_t = max(V_t − sigThre, sigThre)
    else if V_t < Floor_{t-1} then
        Floor_t = max(V_t, sigThre)
        Ceiling_t = min(V_t + sigThre, Ceiling_{t-1})
    else
        Floor_t = Floor_{t-1}
        Ceiling_t = Ceiling_{t-1}
    end if
end procedure
```

- ***Acceptance of a candidate***: The candidate will be accepted and added to *List* when a significant local maximum is found afterwards (e.g., at $t = 1176$ in Table 1). If a local maximum is found when there is no candidate, the maximum is ignored. A local maximum is significant if only if it is found at *Ceiling* while local maxima not found at *Ceiling* will be ignored.

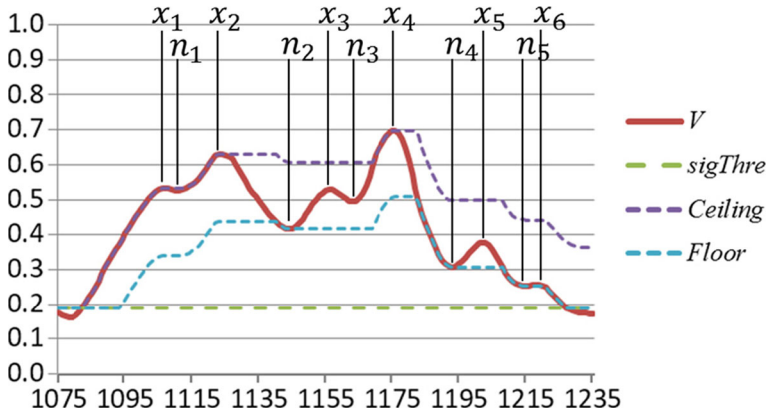

Fig. 7 An example of a moving-period segment

- ***Replacement of a candidate***: If another significant local minimum is found and has a lower value than the current candidate, the timing of the lower one will become a new candidate (e.g., at $t = 1215$ in Table 1).
- ***End of the process***: This process ends when the end of the current moving-period segment is reached, namely, when the velocity curve downwardly crosses the *sigThre* line (e.g., at $t = 1229$ in Table 1). In Fig. 7, n_2 is the only key frame extracted.

Key-frame extraction using both sigThre-crossing and local-minima capturing Algorithm 2 shows all steps for the proposed key-frame extraction. This procedure takes sequential data with F number of frames as its input while its output is a list of key frames. Lines 9 to 10 show the steps for extracting key frames by using the *sigThre*-crossing technique while Lines 11 to 18 show those of the local-minima capturing technique.

Comparison to the conventional velocity-based approach An example on key-frame extraction comparing the two approaches is given in Fig. 8. Only four key frames can be extracted by the conventional velocity-based approach (see vertical bars in the first slot below

Table 1 A break-down analysis of local minima and maxima residing in the moving-period segment in Fig. 7

t	*List*	c	Decision	Reason
1107	–	–	ignore x_1	no candidate
1110	–	–	ignore n_1	insignificant
1123	–	–	ignore x_2	no candidate
1144	–	n_2	promote n_2 to the candidate	significant
1156	–	n_2	ignore x_3	insignificant
1163	–	n_2	ignore n_3	insignificant
1176	n_2	–	accept the candidate	a significant maximum found
1193	n_2	n_4	promote $n_4$to the candidate	significant
1202	n_2	n_4	ignore x_5	insignificant
1215	n_2	n_5	replace the candidate by n_5	a lower minima found
1219	n_2	n_5	ignore x_6	insignificant
1229	n_2	n_5	end of process	cross *sigThre*

the graph). On the other hand, $Mossar_{Splitting}$ can capture 19 key frames (see the second slot below the graph), including those above the *sigThre* line.

Algorithm 2 Generation of a list of key frames

```
1: procedure generateListOfKeyFrames()
2:     List = ∅
3:     c = None
4:     Ceiling_0 = sigThre
5:     Floor_0 = sigThre
6:
7:     for t = 0; t < F; t++ do
8:         updateEnvelope()
9:         if crossSigThreLine() then
10:            List.add(t)
11:        else if V_t > sigThre then
12:            if isLocalMinimum(V_t) and V_t == Floor_t then
13:                if c == None or V_t < V_c then
14:                    c = t
15:                end if
16:            else if isLocalMaximum(V_t) and V_t == Ceiling_t then
17:                if c! = None then
18:                    List.add(c)
19:                    c = None
20:                end if
21:            end if
22:        end if
23:    end for
24:    return List
25: end procedure
```

3.1.2 Graph-based remerging

A set of key frames acquired from the splitting process yields the largest possible number of key frames and thus splits the data file into the most precise level, say, close to the lowest segmentation level (i.e., the Posture Level in Fig. 1). The remerging process produces a number of key-frame sets, each set with decremented number of key frames. Such key-frame sets are used in upper segmentation levels (i.e., the Motion Level and higher levels in Fig. 1). Let $List = \{k_1, k_2, \ldots, k_n\}$ be an initial list of key frames fed from the splitting process to the remerging loop (see Fig. 5); at each iteration, a pair of two consecutive segments that are most similar to each other are merged, and the key frame that previously split them is removed. Algorithm 2 shows the process of remerging where *remeringLoop*() is the main procedure—more explanations are given in the subsections below.

Similarity measurement Let $s_{a,b}$ be a segment starting at key frame k_a and ending at key frame k_b; a list of segments generated by the key frames in *List* can be denoted as $List\_s = \{s_{1,2}, \ldots, s_{n-1,n}\}$ (see Fig. 9). Two additional segments, $s_{f,1}$ and $s_{n,l}$, are then inserted in the beginning

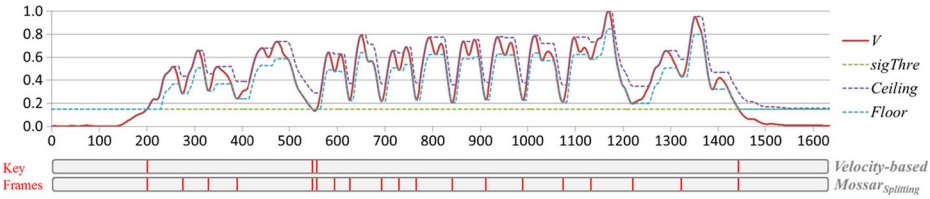

Fig. 8 A comparison on key-frame extraction between a conventional velocity-based method and $\text{Mossar}_{\text{Splitting}}$. This example uses data 85_05 from the CMU database [2]

and the end of *List_s*, respectively, where *f* is the first and *l* is the last frame in a given data file. As discussed in 2.2.1.1, each segment is represented by a graph, and the similarity between each pair of consecutive segments is measured by using a graph kernel. At each iteration, the least important key frame, which is the target for removal, is the key frame whose two segments created by it have the highest similarity.

Remerging Let $Similarity(s_{x-1,x}, s_{x,x+1})$ be the similarity of the most similar pair of consecutive segments, and hence k_x is therefore the targeted key frame for removal. Remerging, as shown in Algorithm 3, may continue until certain conditions for termination are reached. Examples of such conditions can be "the current number of key frames being less than or equal to the expected number α (Line #4 in Algorithm 3)," or "the similarity of the most similar pair of segments (σ_{max}) being smaller than a specified threshold β (Line #15 in Algorithm 3)." On the other hand, such terminal conditions can be omitted when the user interactively selects the best set of key frames on his or her own; namely, in this case, remerging will be processed until all key frames are removed, and then the user picks the set of key frames she or he most prefers. Note that, in actual computation, Algorithm 3 reuses similarity values obtained at previous iterations; for example, at iteration 2, whose input is a set of resulting segments from iteration 1, only the value of $Similarity(s_{2,3},s_{3,4})$ is computed while the values of $Similarity(s_{0,1},s_{1,2})$ and $Similarity(s_{1,2},s_{2,3})$ derived at iteration 1 are reused.

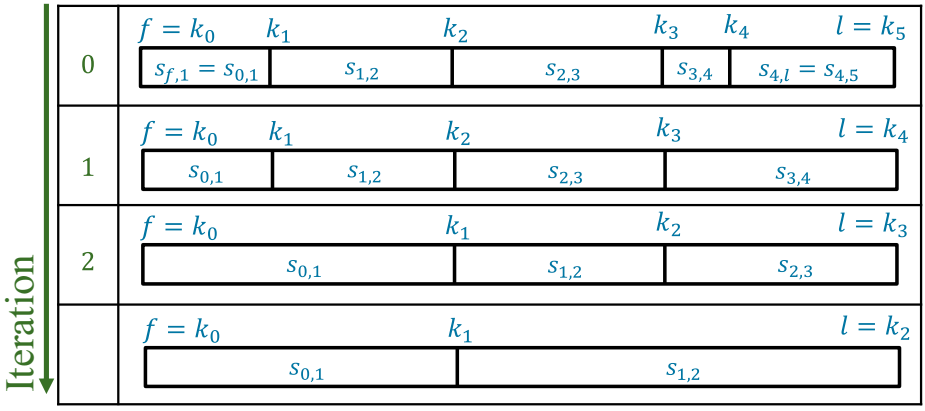

Fig. 9 An example of graph-based remerging where a set of key frames and their resulting segments at the end of each iteration are shown. Iteration 0 shows an initial key-frame set where the subset $\{k_1, k_2, k_3, k_4\}$ are acquired from the splitting process. Note that key frames *f* and *l* are not actual outputs and only used during computation

3.2 Computational complexity

Derivation of the big-O of the whole algorithm starts by considering the splitting process and the remerging process separately. In the splitting process, similarly to the velocity-based approach, a data sequence of interest is simply scanned, and hence its big-O is O(F). In the remerging process, Mossar uses the same methods for graph construction and similarity measurement as in Li and Leung's approach, but we merge key frames provided by the splitting process rather than search for a splitting point in the data sequence. In addition, there is no recursion, and thus the big-O of this part is O($N \cdot \sqrt{N}! \cdot K$) where K is the number of key frames extracted by the splitting process. Although O($N \cdot \sqrt{N}!$) is a big number, it is a constant not related to data size. Because of that, we can simplify O($N \cdot \sqrt{N}! \cdot K$) to O($K$). The summation of two approach in terms of Big-O is O(F) + O(K) or equal to O(F) since $K \leq F$.

Algorithm 3 Remerging process

```
1: procedure remeringLoop()
2:     List ⇐ an initial list of key frames
3:     while List.Count() > 2 do
4:         if List.Count() ≤ α then TERMINATE
5:         end if
6:         σ_max = −1
7:         x = 0
8:         for i = 2; i < List.Count() + 1; i++ do
9:             σ = Similarity(S_(i−2,i−1), S_(i−1,i))
10:            if σ > σ_max then
11:                σ_max = σ
12:                x = i − 1
13:            end if
14:        end for
15:        if σ_max < β then TERMINATE
16:        end if
17:        List.Remove(x)
18:    end while
19: end procedure
```

3.3 Configuration simplicity

There are three main thresholds for configuration of Mossar: *sigThre* for the splitting part, and N and L for the remerging part. In case that data smoothing by moving average is applied, the size of sliding window, *wSize*, will be an additional threshold. The smaller the value of *sigThre* generally increases the number of key frames extracted by the splitting process. Based on the literature, the graph-based approach tends to yield better segmentation performance than the velocity-based approach. Therefore, in order to maximize the performance, the lowest *sigThre* should be assigned to provide the largest possible set of key frames to the remerging process. In the next section on evaluation, we show that the remaining two thresholds, N and L,

inexplicitly affect the performance, data smoothing can be omitted, and, as a result, the three main thresholds can be configured in a simple manner.

4 Evaluation

4.1 Dataset and ground truth

In our experiments, we used the dataset of Subject 86 in the CMU database [2], consisting of 15 motion files, which was previously used by several research groups in their work on motion segmentation (e.g., [4, 6, 12, 25, 29]). Because there are no ground truths[1] provided, each research group created them on their own. In a similar fashion, we first created a ground truth in the lowest level of segmentation hierarchy—motion level (cf., Fig. 1) here because the posture level in the data set contains ambiguous postures—for each motion file, according to observations by 12 subjects as described in the following.

Each subject was tasked to independently create a set of key frames (judgment set) for segmenting a given data file. To obtain a ground truth, kernel density estimation (KDE) [27] was performed based on resulting judgment sets by all subjects. KDE is a non-parametric method for estimating the probability density function of a random variable, and it is remarked as one of the best methods for segmenting one-dimensional data. Key frames were then extracted at maxima of the density curve, during which those maxima with low density were ignored. The type of kernel (e.g., Gaussian, Epanechnikov, or Rectangular) is known to insignificantly affect generation of a ground truth, and we thus empirically selected Gaussian. An example of how a ground truth is generated is shown in Fig. 10. From each motion-level ground truth, a ground truth for the immediate upper level, the motion-cluster level in this case, was then created. Our collection of resulting judgment sets by all subjects and the ground truth for each motion file are publicly available [8].

4.2 Performance metric

We used *Perf* as a metric of performance that was proposed by Ruffieux et al. [20] for ChAirGest 2013, a notable competition on motion segmentation and recognition. It has been used in several motion studies since then [5, 28]. *Perf* is a combination of F_1-*Score* and Accurate Temporal Segmentation Rate (*ATSR*). In other words, the metric considers not only two typically used rates, *Precision* (*P*) and *Recall* (*R*) constituting the F_1-*Score*, but also a frame-level precision in locating key frames.

4.3 Results

In this section, the performance of only the splitting process of Mossar was first benchmarked against a method implemented based on the conventional velocity-based approach mentioned in 2.1.1 (Velocity-based Method). Preprocessing until data smoothing (see 3.1.1.1) was shared between the two approaches. The differences between the two methods are that $\text{Mossar}_{\text{Splitting}}$ adopts normalization, local-minima capturing, and bounding envelope for improvement (see 2.1.4).

[1] ground truth: a set of key frames located at where the data should be partitioned to create segments

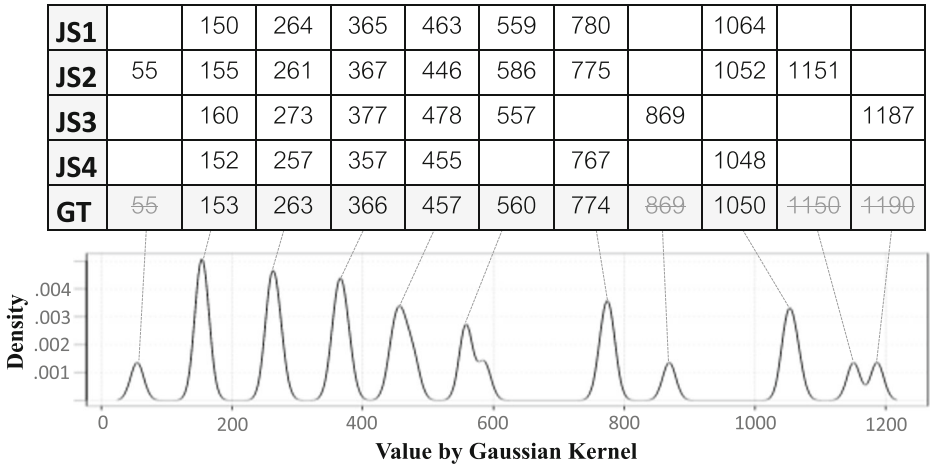

Fig. 10 An example showing how a ground truth (GT) is generated based on judgment sets (JS) from four subjects by using a KDE tool (http://www.wessa.net/rwasp_density.wasp)—a table column illustrates a cluster of up to four key frames (the top four rows)

Then, the performance of the whole method was benchmarked against the graph-based method by Li and Leung [12] (Li & Leung Method). Our evaluation focused on accuracy, configuration simplicity, and computational time. We carried out evaluation of performance in two levels of segmentation hierarchies: the motion-cluster level (Segmentation Level 1 or SL 1) and the motion level (Segmentation Level 0 or SL 0). The former is a commonly used level for the CMU database (e.g., [4, 6, 12, 25, 29]), while the latter is a more precise level where the performance of every motion is counted. Please note that in our approach, performance in the motion level is more crucial because resulting motion-level segments can be remerged in the subsequent process for the motion-cluster level if necessary.

4.3.1 The splitting process

Accuracy The averaged best performances of each of the two methods on all 15 motion files are shown in Table 2. We varied *sigThre* with a precision of 0.01 from 0.01 to 10.00 for Velocity-based Method (un-normalized range in a unit of meters) and from 0.01 to 1.00 for $\text{Mossar}_{\text{Splitting}}$ (normalized range). A moving average was applied for data smoothing. Since a transition between postures sometimes occurs within less than a second in the CMU dataset, a sliding window of 120 frames (or 1 s as the fps of the dataset is 120) tends to be too large to capture such transitions, and, therefore, we tested with different sizes of sliding window, *wSize*, from 1 to 120. For each method, its

Table 2 The averaged best performances of Velocity-based Method and $\text{Mossar}_{\text{Splitting}}$

SL	*Method*	*wSize*	*sigThre*	*Perf*	*ATSR*	*F1*	*Recall*	*Precision*
0	Velocity-based	28.20 ± 27.70	6.77 ± 2.56	0.724	0.616	0.766	0.759	0.816
0	$\text{Mossar}_{\text{Splitting}}$	21.13 ± 24.84	0.09 ± 0.07	0.899	0.801	0.930	0.916	0.952
1	Velocity-based	42.67 ± 46.78	5.44 ± 2.64	0.728	0.656	0.758	0.810	0.748
1	$\text{Mossar}_{\text{Splitting}}$	72.60 ± 40.82	0.36 ± 0.24	0.796	0.739	0.819	0.765	0.919

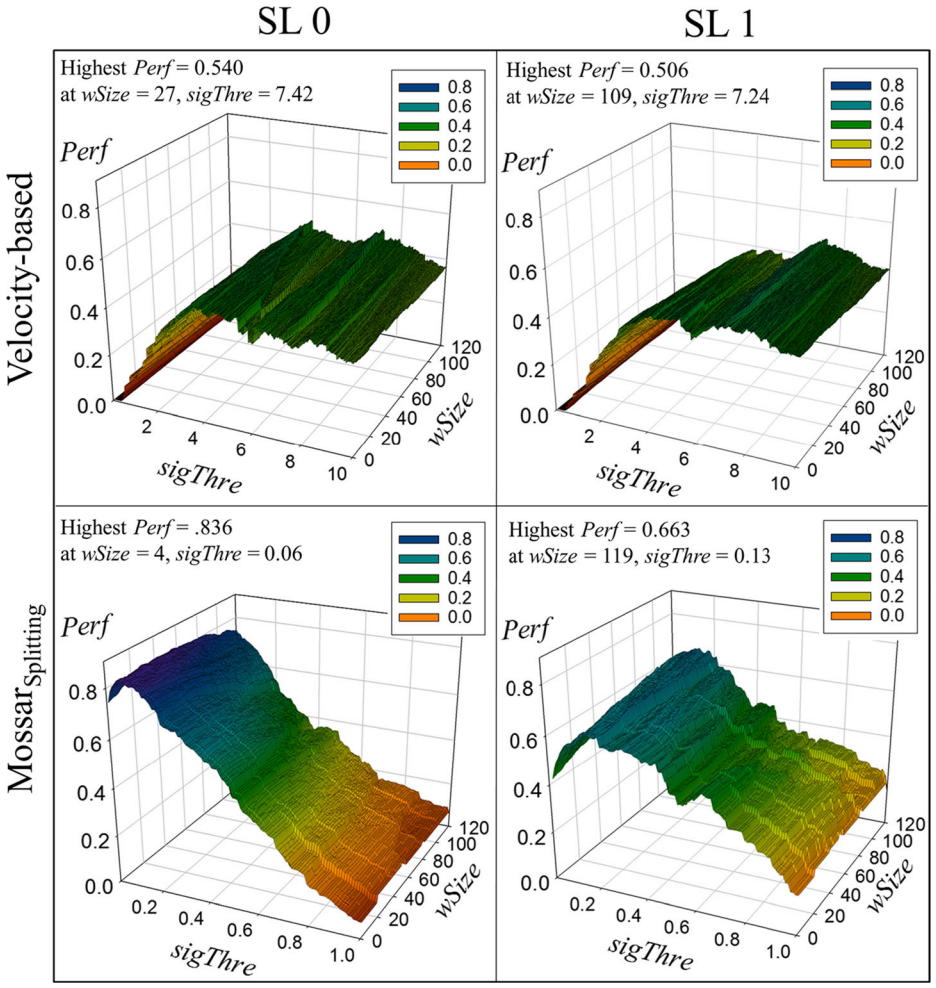

Fig. 11 3D visualization of the performances of Velocity-based Method and Mossar$_{Splitting}$ on 15 files over all combinations of thresholds

performances at all combinations of *sigThre* and *wSize* were tested, and the highest *Perf* was selected for each file; the average of the best performances on 15 files became a record in Table 2. This result demonstrates superiority of our approach in both levels of segmentation hierarchy.

Next, Fig. 11 shows the average performance on 15 files at each combination of *sigThre* and *wSize*, both fixed for all files. Note that this kind of setting is more practical than the previous setting where the thresholds were varied to find the best performance for each file. As can be seen from this figure, the best performance of Mossar is, again, superior to that of Velocity-based Method in both levels. The performance of each method is more sensitive to *sigThre* than *wSize*. In addition, compared to the results in Table 2, the highest performance of Velocity-based Method drastically drops from 0.724 to 0.728 to 0.540 and 0.506 in the motion level and the motion-cluster level, respectively. This indicates that the value of *sigThre* in Velocity-based Method must be properly set

Table 3 The averaged best performances of Li & Leung Method and Mossar

SL	*Method*	*N*	*L*	*Perf*	*ATSR*	*F1*	*Recall*	*Precision*
0	Li & Leung	35.00 ± 5.00	1.93 ± 1.28	0.928	0.785	0.976	0.976	0.976
0	Mossar	10.67 ± 6.51	1.00 ± 0.00	0.957	0.852	0.991	0.982	1.000
1	Li & Leung	29.67 ± 10.26	2.07 ± 1.28	0.980	0.942	0.991	0.983	1.000
1	Mossar	10.33 ± 7.90	1.40 ± 1.12	0.977	0.952	0.984	0.969	1.000

for each file in a given dataset in order to achieve a high performance. On the other hand, the highest performance of Mossar degrades more gracefully, in particular, in the motion level which as mentioned earlier is more important to this approach.

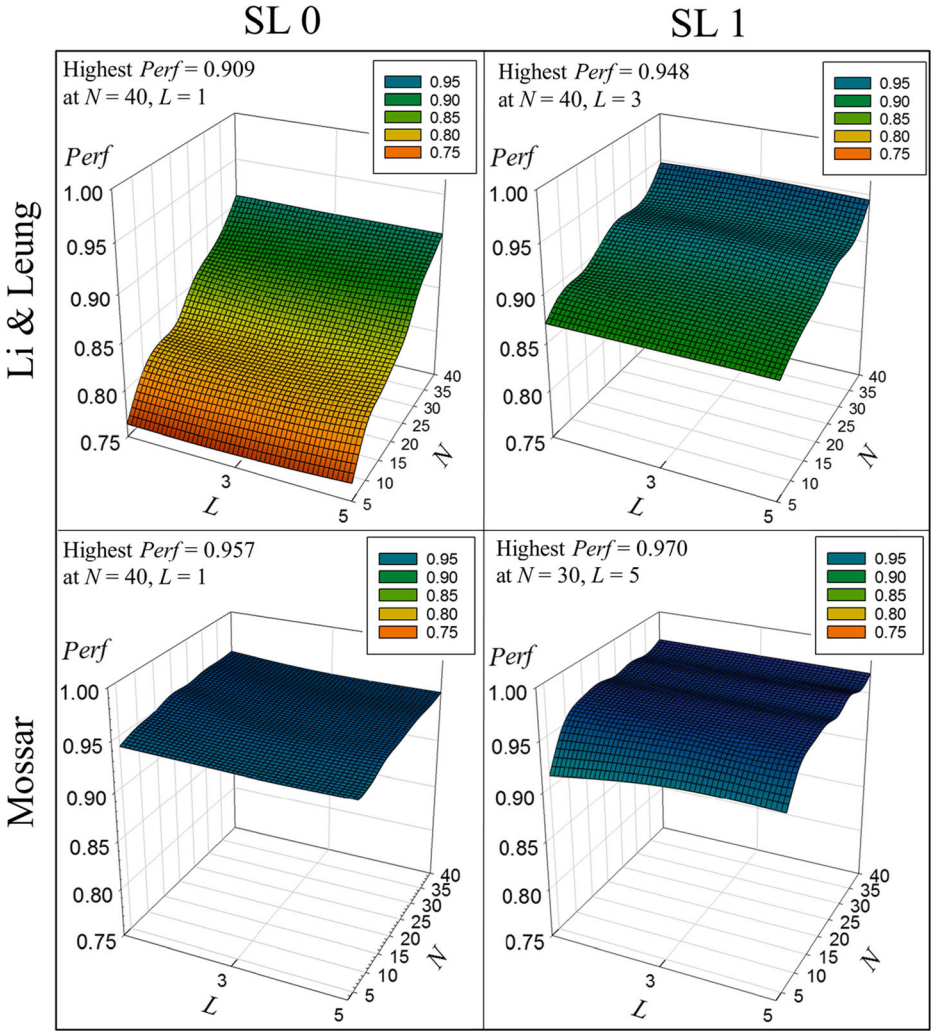

Fig. 12 3D visualization of the performances of Li & Leung Method and Mossar on 15 files over all combinations of thresholds

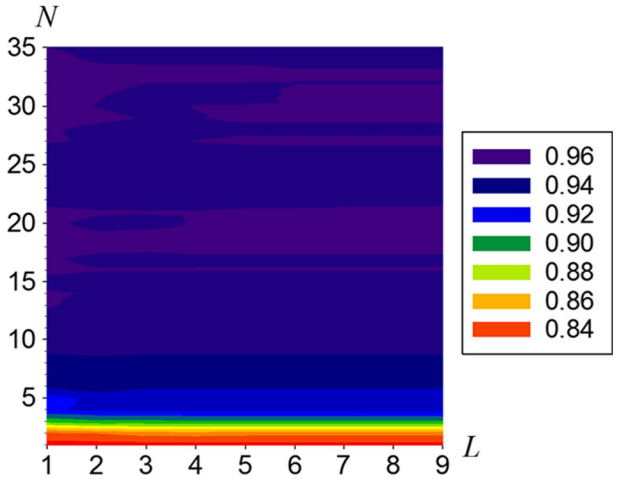

Fig. 13 The averaged performances of Mossar over all combinations of threshold values

Configuration simplicity In all four cases in Fig. 11, the performance is mainly affected by *sigThre*, but slightly affected by *wSize*. We explored the effects of threshold setting to the performance of Mossar in the motion level. In this level, high values of *wSize* were found unpreferable. With a moving average, the highest *Perf* of 0.836 was found at a combination of *wSize* = 4 and *sigThre* = 0.06, while without a moving average (*wSize* = 1), the highest *Perf* of 0.835 was found at *sigThre* = 0.06. Since the difference in performance is very small, we consider that data smoothing can be omitted for the sake of configuration simplicity.

More importantly, when the remerging process is added, the first priority should be given to minimization of missing key frames, i.e., maximizing the number of input key frames to be provided to the remerging process. Because the number of resulting key frames in general decreases when *sigThre* increases, *sigThre* can be set to 0.01 for providing the largest possible set of key frames to the remerging process. In summary, *wSize* can be set to 1, because of no need for data smoothing, and one can set *sigThre* to 0.01 as a rule-of-thumb.

In addition, reasons on why the velocity-based method shows an opposite trend with $\text{Mossar}_{\text{splitting}}$ as the value of *sigThre* increases can be explained as follows. Besides *sigThre*-crossing, $\text{Mossar}_{\text{splitting}}$ reuses *SigThre* for local-minima capturing; the gap

Table 4 Average performances of Li & Leung Method in both SLs

Perf		*L*			
		1	3	5	AVG
N	**5**	0.817	0.817	0.819	0.818
	10	0.846	0.846	0.847	0.846
	15	0.855	0.856	0.857	0.856
	20	0.864	0.865	0.867	0.865
	25	0.885	0.884	0.882	0.884
	30	0.892	0.895	0.897	0.895
	35	0.903	0.907	0.908	0.906
	40	0.928	**0.928**	0.926	0.927
	AVG	0.874	0.875	0.876	0.875

The best performance are in bold

Table 5 Average performances of Mossar in both SLs

Perf		*L*			
		1	3	5	AVG
N	**5**	0.926	0.931	0.931	0.929
	10	0.954	0.954	0.953	0.953
	15	0.957	0.955	0.955	0.955
	20	0.961	0.957	0.957	0.958
	25	0.958	0.956	0.956	0.957
	30	0.961	0.958	0.960	0.960
	35	0.960	0.956	0.956	0.958
	40	**0.964**	0.960	0.960	0.961
	AVG	0.955	0.953	0.954	0.954

The best performance are in bold

between *Ceiling* and *Floor* of the bounding envelope is generally equal to *sigThre* when *sigThre* ≥ *Floor* (see Fig. 7). The method decides that minima are significant only if they are found at *Floor* of the envelope. This means when *sigThre* is small, minima will be easily considered significant. SL 0 is a highly-precise segmentation level in which most stages of movement are expected to be captured, and this is why preferable results are on low values of *sigThre*. On the other hand, for velocity-based, when *sigThre* is very low, key frames will not be captured well, as the velocity curve hardly crosses a given significant threshold line.

Computational time A computer with Intel(R) Core(TM) i7-3612QM CPU @ 2.10GHz processor was used for testing computation time. We separated preprocessing from segmentation because the former was shared between the two approaches in this experiment. Preprocessing until data smoothing (3.1.1.1) was tested 1.2 × 1E5 times, and the average time for preprocessing 15 files was 1680.515 milliseconds—this number of times was from our experiment for Velocity-based Method in Fig. 11 having 120 *wSize*s × 1000 *sigThre*s, leading to 1.2 × 1E5 tests. The segmentation process itself costs much less processing time: on average 5.775 and 9.031 milliseconds for Velocity-based Method and $\text{Mossar}_{\text{Splitting}}$, respectively. In terms of total processing time, combining preprocessing and segmentation, $\text{Mossar}_{\text{Splitting}}$ costs only 0.19% more.

Table 6 The total processing time in seconds of Li & Leung Method on 15 files (*N* of 1 cannot be tested, details of which are given in 5.1)

Time		*L*				
		1	3	5	AVG	hh:mm:ss
N	5	2617.240	2536.780	2520.233	2558.084	00:42:38
	10	2063.009	1889.216	1809.758	1920.661	00:32:00
	15	2042.511	1836.024	1555.346	1811.293	00:30:11
	20	2071.770	1835.115	1715.124	1874.003	00:31:14
	25	2760.651	2346.955	2240.179	2449.262	00:40:49
	30	6499.946	6363.356	4965.889	5943.064	01:39:03
	35	25,025.441	23,919.123	19,796.086	22,913.550	06:21:53
	40	113,847.044	108,814.124	90,057.388	104,239.518	28:57:19

Table 7 The total processing time in seconds of Mossar on 15 files

Time		*L*				
		1	3	5	AVG	hh:mm:ss
N	1	64.745	64.951	65.285	64.994	00:01:04
	5	149.021	150.790	154.207	151.339	00:02:31
	10	147.481	150.525	154.014	150.673	00:02:30
	15	155.244	158.705	159.238	157.729	00:02:37
	20	170.738	175.415	178.800	174.984	00:02:54
	25	291.488	293.236	302.426	295.717	00:04:55
	30	896.441	900.379	905.571	900.797	00:15:00
	35	3691.799	3852.517	3946.443	3830.253	01:03:50
	40	16,189.141	16,426.283	16,146.482	16,253.969	04:30:53

4.3.2 Combination of both splitting and remerging processes

Accuracy To evaluate the performance of Mossar for both splitting and remerging processes, we fed the largest set of key frames in our setting, extracted by the splitting process with $sigThre = 0.01$, as input to the remerging process. As done in Section 4.1.3.1, the averaged best performance of Mossar and Li & Leung Method on all 15 motion files are shown in Table 3, where two main thresholds *N* and *L* were varied to find the best *Perf* for each file. For both methods, we varied *N* with a precision of 5 from 5 to 40 and *L* with a precision of 2 from 1 to 5. The results in this table show that Mossar has as high performance as Li & Leung Method.

Next, the average performance on 15 files at each combination of *N* and *L*, both fixed for all the files, is shown in Fig. 12. In comparison to that of Li & Leung Method, the performance of Mossar appears less sensitive to changes in values of the two thresholds throughout all parameter space in the motion level and for $N \geq 10$ in the motion-cluster level. This indicates Mossar's configuration simplicity.

In addition, since Mossar's user can interactively control the number of key frames, enabling her or him to obtain key frames in an arbitrary level of hierarchy, the average performance of Mossar in the two segmentation levels shown in Fig. 13 can be heuristically used as a good indicator for the performance in such an arbitrary level. Tables 4 and 5 show the

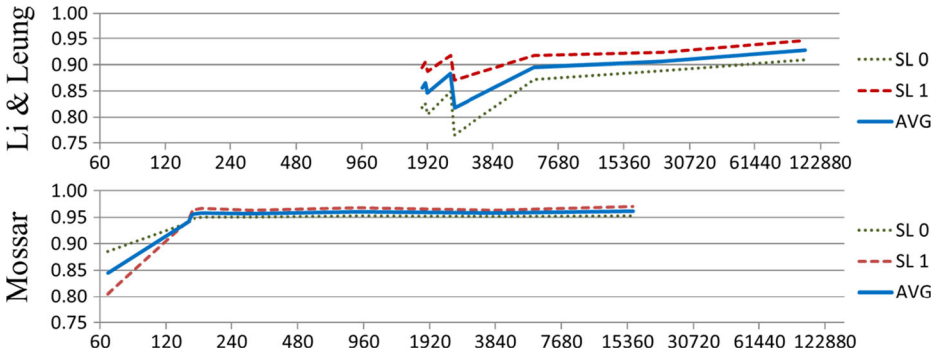

Fig. 14 The relationship between performance and time. The horizontal axis represents time for processing 15 files in seconds (logarithmic scale) while the vertical axis shows *Perf*

averaged performances of Li & Leung Method and Mossar, respectively. The highest *Perf* of Mossar is 0.964 with $L = 1$ and $N = 40$, indicating that this pair should be used during interactive control.

Configuration simplicity It is clear to see that the performance of Li & Leung Method depends on N whose increase in value results in rapid growth in processing time. According to our further finding by testing combinations of thresholds at a precision of 1, as shown in Fig. 13, the performance of Mossar becomes stable between 0.940 and 0.960 when the value of N is greater than 6. Note that when using techniques for graph construction and similarity measurement by Li and Leung, too small values of N are generally not recommended. Nevertheless, for Mossar, it is still possible to set N to 1, by which the processing time is minimized and its performance is 0.845.

Computational time The total processing time on 15 files for each method is shown in Tables 6 and 7. For both methods, their processing time drastically increases from N above 30. By using results in Fig. 12 and Tables 6 and 7, the relationship between computational time and performance was examined, and the results are shown in Fig. 14. According to this figure, the performance of Li & Leung Method tends to increase when a bigger time budget is given (except for very small values of N), while the performance of Mossar is relatively stable. We, therefore, suggest that optimization of thresholds should take a trade-off between performance and time complexity into consideration. As shown earlier in Table 5, the highest *Perf* of Mossar is 0.964 with $L = 1$ and $N = 40$, but this performance costs 4 h and a half of computation. On the other hand, the performance of 0.961 with $L = 1$ and $N = 20$ costs only about 3 min.

5 Conclusions and discussions

Our experiments demonstrated outstanding performance of Mossar in terms of both computational time and segmentation accuracy. The design of Mossar took flexibility in diverse needs for segmentation into account. Mossar was then created by hybridization of two existing approaches, which enables reduction in the number of thresholds in use and facilitation of configuring their values. Important issues regarding potential advantages stated in the beginning of Section 3 are highlighted and discussed below.

5.1 Fast and highly accurate segmentation

Through the experiments, it was confirmed that our aim to create a method that yields high segmentation accuracy as the graph-based method but with low computational complexity as the velocity-based method was fulfilled—the accuracy was compared in Fig. 12 while the big-O was discussed in 3.2. The remerging part of Mossar adopts techniques for graph construction and similarity measurement from the graph-based approach, but our results in Section 4.3.2 show that the thresholds used in Mossar become more robust. Mossar can use N set to 1 that minimizes the processing time while the graph-based method practically cannot. This can be explained as even though both approaches use the same graph representation and kernels, Li & Leung Method

finds segmenting points by minimizing the similarity between two segments while Mossar merges segments by maximizing the similarity. When N is too low, Li & Leung Method faces a problem because many pairs of segments share no matched edge. This leads to the similarity of zero; as a result, the number of inserted key frames as well as search iterations and recursions tend to be maximized, resulting in degraded performance and very high computational complexity. On the other hand, Mossar is more tolerant to such cases as segments with no matched edges are the latest to be processed. Due to the robustness in terms of threshold values, N, which highly affects to the time complexity (see Table 6 and 7), can be configured with a low value, which drastically reduces the time complexity.

For how the number of frames (F) or key frames (K) affects the computation complexity of the remerging process. If the number of key frames is high, the computation complexity will only grow linearly, i.e., O(K). The worst case of K is that of $K = F$. However, from the experiments, when *sigThre* is set to 0.01 for obtaining the large set of key frames, the average ratio of F to K is 354:1, and the worst ratio is 148:1—since the frame rate is 120 fps, the ratio of 354:1 means a key frame is found every about three seconds, while 148:1 means a key frame is found almost every second.

5.2 Flexibility to diverse needs for segmentation

Different applications require different levels of segmentation hierarchies (Fig. 1). In most existing approaches, it is unavoidable to adjust thresholds and re-run the segmentation process to obtained key frames for each specific level of hierarchy. However, Mossar allows the user to interactively adjust the number of key frames to match her or his desired level of segmentation hierarchy, and sets of key frames for all hierarchies can be obtained by only one segmentation process (discussed in 3.2.2).

5.3 Configuration simplicity

Our design of Mossar focuses on minimizing the number of thresholds and eliminating the need to specify their values. Four thresholds derived from existing approaches were examined in this work:

- ***wSize***: a window size of moving average used for data smoothing in the conventional velocity-based approach. We found that data smoothing only slightly affected the performance of Mossar and can be omitted (see 4.3.1). This can be explained as that Mossar embeds a concept of minimal bounding envelope by which capturing of insignificant minima can be avoided without data smoothing.
- ***sigThre***: the main threshold from the velocity-based approach. For the velocity-based splitting part of Mossar, the value of this threshold can be set to 0.01 as a rule of thumb (see 4.3.1). This value will produce the largest possible set of key frames to be fed as input to the remerging process.
- ***N and L***: after eliminating unnecessary thresholds, these two are the remaining thresholds Mossar inherits from the graph-based approach (see 2.2.3). Our results showed that the two thresholds became more robust; by avoiding too small values of N, a stable performance can be obtained (see 4.3.2). Our experiments also demonstrated a possibility to optimize threshold values for an arbitrary level of hierarchy.

In terms of the trade-off between performance and processing time, the velocity-based splitting part of Mossar is equivalent to an improved velocity-based approach; in our experiments, it yielded better performance with an insignificant increase in computational time (see 4.3.1). In comparison to the graph-based approach, while the performance of Li & Leung Method tended to increase when more time budget is given, the performance of Mossar was relatively stable (see Fig. 14). Except for cases where a very high segmentation performance is demanded regardless of cost, we suggest Mossar as a candidate motion segmentation method for its feasibility in terms of computational efficiency and configuration simplicity.

Acknowledgements The authors wish to thank Professor Kingkarn Sookhanaphibarn, Bangkok University, for her assistance in the technical editing of the manuscript. We would also like to express special thanks to the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

1. Bulut E, Capin T (2007) Key-frame extraction from motion capture data by curve saliency. In: Computer animation and social agents 2007 Jun (pp 119–123)
2. Carnegie Mellon University. CMU graphics lab motion capture database. http://mocap.cs.cmu.edu. Accessed 1 Jan 2018
3. Choensawat W, Nakamura M, Hachimura K (2015) GenLaban: a tool for generating Labanotation from motion capture data. Multimed Tools Appl 74(23):10823–10846
4. Devanne M, Wannous H, Pala P, Berretti S, Daoudi M, Del Bimbo A (2015) Combined shape analysis of human poses and motion units for action segmentation and recognition. In: Automatic face and gesture recognition (FG), 2015 11th IEEE international conference and workshops on 2015 May 4 (Vol. 7, pp 1–6). IEEE
5. Escalera S, Athitsos V, Guyon I (2016) Challenges in multimodal gesture recognition. J Mach Learn Res 17(72):1–54
6. Gong D, Medioni G, Zhu S, Zhao X (2012) Kernelized temporal cut for online temporal segmentation and recognition. Comput Vis–ECCV 2012:229–243
7. Hachimura K, Nakamura M (2001) Method of generating coded description of human body motion from motion-captured data. In: Robot and human interactive communication. Proceedings. 10th IEEE International Workshop on 2001 (pp 122–127). IEEE
8. ICE Lab Mossar. https://sites.google.com/site/icelabuki/mossar. Accessed 1 Jan 2018
9. Keogh E (2002) Exact indexing of dynamic time warping. In: Proceedings of the 28th international conference on very large data bases 2002 Aug 20 (pp 406–417). VLDB Endowment
10. Khan S, Bailey DG, Gupta GS (2014) Pause detection in continuous sign language. Int J Comput Appl Technol 50(1–2):75–83
11. Krüger B, Vögele A, Willig T, Yao A, Klein R, Weber A (2017 Apr) Efficient unsupervised temporal segmentation of motion data. IEEE Trans Multimed 19(4):797–812
12. Li M, Leung H (2016) Graph-based representation learning for automatic human motion segmentation. Multimed Tools Appl 1;75(15):9205–9224
13. Li M, Leung H, Liu Z, Zhou L (2016) 3D human motion retrieval using graph kernels based on adaptive graph construction. Comput Graph 54:104–112
14. Lim IS, Thalmann D (2001) Key-posture extraction out of human motion data. In: Engineering in medicine and biology society. Proceedings of the 23rd annual international conference of the IEEE 2001 (Vol. 2, pp 1167–1169). IEEE
15. Liu F, Zhuang Y, Wu F, Pan Y (2003) 3D motion retrieval with motion index tree. Comput Vis Image Underst 92(2):265–284
16. Miura T, Mitobe K, Yukawa T, Kaiga T, Taniguchi T, Tamamoto H (2011) Adaptation of grouping structure analysis in GTTM to hierarchical segmentation of dance motion. Inf Media Technol 6(1): 172–192

17. Northwestern University Process Optimization Open Textbook Spatial branch-and-bound. https://optimization.mccormick.northwestern.edu/index.php/Spatial_branch_and_bound_method. Accessed 1 Jan 2018
18. Paliyawan P, Sookhanaphibarn K, Choensawat W, Thawonmas R (2015) Towards universal kinect interface for fighting games. In Consumer Electronics (GCCE), 2015 I.E. 4th Global Conference on 2015 Oct 27 (pp 332–333). IEEE
19. Panagiotakis C, Holzapfel A, Michel D, Argyros AA (2013) Beat synchronous dance animation based on visual analysis of human motion and audio analysis of music tempo. In: International symposium on visual computing 2013 Jul 29 (pp 118–127). Springer Berlin Heidelberg
20. Ruffieux S, Lalanne D, Mugellini E (2013) ChAirGest: a challenge for multimodal mid-air gesture recognition for close HCI. In: Proceedings of the 15th ACM on International conference on multimodal interaction 2013 Dec 9 (pp 483–488). ACM
21. Shiratori T, Nakazawa A, Ikeuchi K 2018 Detecting dance motion structure using motion capture and musical information. In: Proc. international conference on virtual systems and multimedia (VSMM) 2004 Nov (Vol. 3)
22. Tunca C, Pehlivan N, Ak N, Arnrich B, Salur G, Ersoy C (2017) Inertial sensor-based robust gait analysis in non-hospital settings for neurological disorders. Sensors 17(4):825
23. Van Mieghem P (2010) Graph spectra for complex networks. Cambridge University Press, Cambridge ISBN-13: 9781107411470
24. Vlachos M, Hadjieleftheriou M, Gunopulos D, Keogh E (2003) Indexing multi-dimensional time-series with support for multiple distance measures. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining 2003 Aug 24 (pp 216–225). ACM
25. Vögele A, Krüger B, Klein R (2014) Efficient unsupervised temporal segmentation of human motion. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2014 Jul 21 (pp 167–176). Eurographics Association
26. Wang Q, Kurillo G, Ofli F, Bajcsy R (2015) Unsupervised temporal segmentation of repetitive human actions based on kinematic modeling and frequency analysis. In: 3D Vision (3DV), 2015 International Conference on 2015 Oct 19 (pp 562–570). IEEE
27. Wessa P. Kernel density estimation (v1.0.12) in free statistics software (v1.1.23-r7). http://www.wessa.net/rwasp_density.wasp. Accessed 1 Jan 2018
28. Yin Y, Davis R (2013) Gesture spotting and recognition using salience detection and concatenated hidden markov models. In: Proceedings of the 15th ACM on international conference on multimodal interaction 2013 Dec 9 (pp. 489–494). ACM
29. Zhou F, De la Torre F, Hodgins JK (2013) Hierarchical aligned cluster analysis for temporal clustering of human motion. IEEE Trans Pattern Anal Mach Intell 35(3):582–596

Pujana Paliyawan is currently a Ph.D. student in the Department of Human and Computer Intelligence, College of Information Science and Engineering, Ritsumeikan University, Japan. His research interests include motion analysis, artificial intelligence, machine learning, and human computer interaction.

Worawat Choensawat received his Dr. Eng degree from School of Science and Engineering, Ritsumeikan University in 2012. During his stay in Ritsumeikan University, he was a research assistant in the Global COE program of Digital Humanities Center for Japanese Arts and Cultures. Currently he is an assistant professor in the School of Information Technology and Innovation, Bangkok University, Thailand. His current research is multimedia applications for human body analysis and other education purposes.

Ruck Thawonmas is a full professor at the College of Information Science and Engineering, Ritsumeikan University. His research interests include game AI and computational intelligence.