3D Res (2018)9:5
https://doi.org/10.1007/s13319-018-0158-y

3DR EXPRESS

Master of Puppets: An Animation-by-Demonstration Computer Puppetry Authoring Framework

Yaoyuan Cui · Christos Mousas

Received: 15 December 2017 / Accepted: 21 January 2018

Abstract This paper presents Master of Puppets (MOP), an animation-by-demonstration framework that allows users to control the motion of virtual characters (puppets) in real time. In the first step, the user is asked to perform the necessary actions that correspond to the character's motions. The user's actions are recorded, and a hidden Markov model is used to learn the temporal profile of the actions. During the runtime of the framework, the user controls the motions of the virtual character based on the specified activities. The advantage of the MOP framework is that it recognizes and follows the progress of the user's actions in real time. Based on the forward algorithm, the method predicts the evolution of the user's actions, which corresponds to the evolution of the character's motion. This method treats characters as puppets that can perform only one motion at a time. This means that combinations of motion segments (motion synthesis), as well as the interpolation of individual motion sequences, are not provided as functionalities. By implementing the framework and presenting several computer puppetry scenarios, its efficiency and flexibility in animating virtual characters is demonstrated.

Keywords Computer puppetry · Performance animation · Character animation · Motion control · HMM

1 Introduction

The recent increase in affordable motion capture and sensing interfaces (i.e., Kincet, Wiimote, and Leap) permits users to interact with displayed and virtual content in more complex and advanced ways, thus allowing them to become more immersed in the virtual experience. Such devices can capture the full-body's movements or simply finger gestures or facial expressions. Based on previously published work [1], it can be said that the ability to interact with a virtual environment through body movements enhances the user experience.

Such interfaces, have been used to control virtual characters as well as to allow users to navigate and interact with virtual environments. So-called computer puppetry or performance animation methods are mainly used for controlling characters, and the entertainment, gaming, and virtual reality industries

A1 **Electronic supplementary material** The online version of
A2 this article (https://doi.org/10.1007/s13319-018-0158-y) con-
A3 tains supplementary material, which is available to authorized users.

A4 Y. Cui · C. Mousas (✉)
A5 Graphics and Entertainment Technology Lab, Department
A6 of Computer Science, Southern Illinois University,
A7 Carbondale, IL 62901, USA
A8 e-mail: christos@cs.siu.edu

A9 Y. Cui
A10 e-mail: yaoyuan.cui@siu.edu

Published online: 03 February 2018

generally benefit from them. The main disadvantage of most performance-driven character control approaches is their limited ability to work only with human-like virtual characters. However, in computer puppetry, users should be able to interact with a variety of characters and creatures, even those that are not humanoid, based on body movements, finger gestures, and facial expressions.

In the skeletal-based, performance-driven animation of humanoid characters, the user's joints are mapped to the joints of a virtual character. However, in some cases, the target character may not be a humanoid. For example, it might be a virtual creature or an object, such as a table or a chair, that needs to be animated. When such complex non-human characters and objects need to be animated, it is necessary to define advanced mapping techniques, which are generally called retargeting [2]. It should be noted that retargeting techniques are responsible for transferring the motion that belongs to a particular morphology to a different one. The disadvantage of such retargeting techniques is mainly the time-consuming manual work that is required to define the correspondence between the user's poses and the non-human character's poses.

This paper focuses on the user's action-based control of a virtual character's motion. Specifically, it addresses the issue of controlling an individual character's independent motion by a particular action performed by a user. Such character control mechanism can also be described as a highly constrained computer puppetry control method. This means that the virtual puppet is able to perform only specific motions as well as one motion at a time, and the performers can use their bodies or body parts (e.g., fingers and face) to control the motions independently. To solve this constrained problem, a hidden Markov model (HMM) that learns the temporal profile of the user's actions is used. During the runtime of the applications, the real-time prediction is achieved using the forward algorithm, which predicts the action of the user and consequently the motion of the character that should be displayed as well as the evolution of the character's motion based on the evolution of the user's actions.

In the Master of Puppets (MOP) framework, it is not necessary to perform exhaustive mapping between the user's body parts and the character's motions. This is achieved by directly associating the time progression of the user's action with the time progression of the character's motion. Therefore, a generalized soft mapping process is used in this approach compared to other methodologies that highly constrain this process. The generalization process for controlling virtual characters makes it possible for the user to independently control the motions of different characters (humanoid or non-humanoid), even if the target characters are dissimilar or have different morphological variations compared with the user. To summarize, the MOP framework can be characterized as follows:

- *Easy to Use* Users are not required to manually define the correspondence between their joints and the character's joint angles or control points. Each user captures their actions and assigns them to the corresponding animation of the virtual character.
- *Fast* The framework learns the temporal profile of each action only a few seconds after capturing the actions of the user.
- *Flexible* The framework provides the ability to animate the target characters without the need for morphological similarities to a human and without the need to use only animations of skeletal-based characters.

This remainder of this paper is organized as follows. Section 2 discusses previous work that is related to the proposed methodology. Section 3 presents the core part of the MOP framework, and Sect. 4 discusses the implementation details, examples, and evaluation of the presented framework. Finally, Sect. 5 provides conclusions and possibilities for future work.

2 Related Work

Different research fields can include the MOP framework. Therefore, this section outlines previous work related to MOP and discusses the contributions of the presented framework.

2.1 Input Devices

Interactive character control mechanisms can be classified based on the devices used to animate virtual characters [1]. The keyboard and mouse are the most common ways to control the motions of virtual characters, as well as joysticks, which are used in

most game consoles [3]. It is also important to note that more specialized control mechanisms have been introduced to animate virtual characters, such as text input [4, 5], speech-based [6], and sketch-based interfaces [7]; RGB-D [8, 9] and IR [10] sensors; data gloves [11] or color gloves [12]; simple accelerometers [13, 14]; and even 3D-printed custom interfaces [15]. MOP allows the use of different motion sensing devices to control a virtual character. In its current version, the MOP framework supports the use of Microsoft's Kinect and Leap motion devices. The flexibility of handling input from more than one device simultaneously is another unique feature of MOP. (Sect. 4.1.2 and the video that accompanies this paper present an example in which more than one device is used simultaneously for capturing the user's activity.)

2.2 Character Animation

Characters can be animated in several different ways, such as based on the keyframing method, which is characterized by its time-consuming process as well as by the specialization the animator should have. Computational methods use existing motion data (data-driven methods) and adapt the motion to fulfill the necessary constraints of the virtual environment. Among the numerous data-driven motion synthesis techniques that exist, interpolation [16], blending [17], splicing [18, 19], and warping [20] are used most extensively to make existing motion data reusable. A variety of statistical and dimensionality reduction methods have been used in the past to handle data efficiently. Examples include principal component analysis (PCA) [21], Gaussian process latent variable models (GPLVM), kernel methods [22, 23], and multi-dimensional scaling [24] methods. Such methods have been used to parameterize the existing motion data to animate virtual characters in a proper way [25]. MOP can be characterized as a data-driven character animation framework, since the characters are animated using motion data. However, MOP does not consider any motion synthesis technique in its current form. It only provides users the ability to control the evolution of motions that are assigned to virtual characters.

2.3 Performance Animation

In performance animation, users control a virtual character with their bodies. Motion capture systems usually provide the necessary input signals retrieved either from accelerometers [14] or from optical input [26]. Then, techniques based on kinematics-related solutions [27] or data-driven motion reconstruction [28, 29] are used to ensure that the reconstructed motion lies in the pose space of the existing motion data. It should be noted that data-driven techniques can reconstruct natural-looking poses using a very small number of control inputs. The reduction of the error is achieved due to the prior knowledge that is obtained from the example data. The use of one [30], two [31], and six [28] input(s) has also been examined. It should be noted that statistical models based on large motion datasets [26, 28] are used to build prior knowledge to achieve such an under-constrained reconstruction process. However, even if the use of one [30] or two [31] input(s) provides a realistic motion, complex motions cannot be reconstructed, especially when the user performs an action with a body part that is not captured. MOP relates to performance animation since users must use their bodies or body parts to control the motion of virtual characters.

2.4 Computer Puppetry

In computer puppetry, users control the motion of a virtual character not necessarily by defining a joints-to-joints mapping. Mid-air gestures and machine learning techniques have also been used to define correspondence between the input actions of a user and the output motions of a character. In most computer puppetry methods, it is quite important to choose the right input device [32]. Dontcheva et al. [33] proposed a method that does not require the time-consuming manual selection of a character's body parts. Instead, this is achieved by aligning the temporal sequences using canonical correlation analysis. Seol et al. [34] proposed a computer puppetry approach based on a correlation mapping process of the features of a user's actions and a character's motions. The approach developed by Chen et al. [9] maps joint positions between the user and the object. The animation is achieved using mesh deformation techniques. Vögele et al. [35] developed a method in

which the motions of two users are mapped in a variety of quadrupedal characters. Yamane et al. [36] proposed a method that works offline and performs mapping between human motion and non-human motion. To achieve natural-looking results, the user should manually define 30 to 50 correspondences. Finally, hybrid controllers [35, 37] that allow the user to navigate and interact with virtual objects in virtual environments in a more sophisticated way through mid-air gestures and the direct manipulation of body parts have been also introduced. MOP differs from previously developed puppetry approaches, since users can control the time progression of the motion instead of the motion itself, and the hard-constrained piece-by-piece mapping process is not applicable.

2.5 Score Following

The MOP framework is inspired by existing work that examines "score following" or "gesture following". Such methods are responsible for learning the temporal profile of a continuous stream of data. Various applications benefit from score following, since such methods help improve the way users interact with computers. Real-time music accompaniment [38] and the gestural control of sound [39] are good examples of this. The most common approaches for score following include the use of dynamic time warping (DTW) [40], HMM [41], neural networks (NN) [42], support vector machines (SVM) [43], and dynamic programming (DP) [44]. The score following method developed in this paper is based on HMM.

2.6 Contribution

The MOP framework contributes several unique features. First, the correspondence between a user's actions and a character's motions are built automatically even if the input parameters are not directly related to the target parameters. Second, the HMM allows users to control the evolution of a character's motion based on their own activities. Given this, the output motion is not synthesized but rather displayed just as it is. Third, the flexibility of the proposed method to handle different characters as well as a variety of inputs are two additional important advantages. MOP automatically handles the data and allows the user to interact immediately with the virtual character. The examples in this paper show that without any additional effort, MOP provides users the ability to control more than one character simultaneously, and it allows more than one user to control a target character. It is assumed that the contributions provided by the proposed method could benefit the entertainment community, since all the functionalities implemented within a single framework can be used for a variety of computer puppetry interaction scenarios. For example, data-driven digital shadow puppets, which are a good example of animating constrained virtual puppets using specific actions and corresponding output motions, could benefit from MOP.

3 Methodology

This section presents the key components of the MOP framework. Specifically, it describes how users are able to control virtual characters based on their actions. The architecture of the MOP framework, which is divided into three parts (pre-processing, HMM construction and training, and runtime character control) is illustrated in Fig. 1.

3.1 Representation of Motion Data

MOP requires at least two motion sequences. The first one is the user's action, and the second is the motion that animates the virtual character. The first motion is captured using one of the compatible sensors. The user designs the second set, or if a humanoid character is being animated, it can be captured. The rest of this section describes how both types of data are represented.

3.1.1 User

The user's action is represented as $X_{1:T} = [x_1, \ldots, x_T]$, where T denotes the total number of frames. Each x_t contains the pose of the user at time t, which can also be represented as rotations of joint angles r, such as $x_t = [r_t(1), \ldots, r_t(L)]$, where $x_t \in \mathbb{R}^{dx}$ and L denote the number of captured joints of the user. It should be noted that the captured position of the user's root is not included. It is assumed that the virtual character will remain in the same position in the virtual environment. The same representation is used when capturing either the full-body motion or the finger motion of a user.

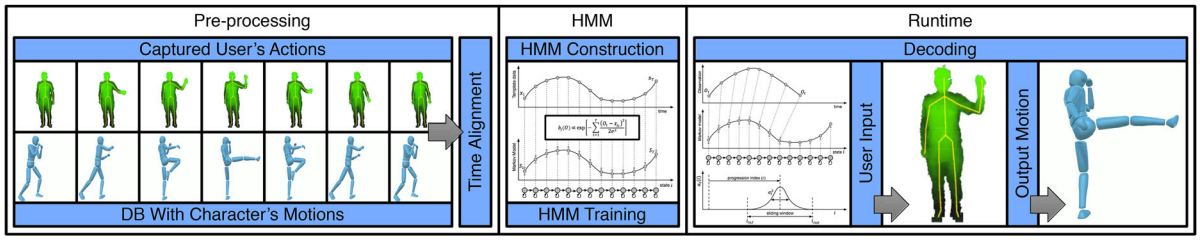

Fig. 1 The architecture of MOP framework

However, the input parameters can be also retrieved from the face of the user. In this case, the user's motion is represented by control vectors $x_t = [p_t(1), \ldots, p_t(G)]$ where G denotes the total number of captured features of the user's face at time t. It should be noted that each $p_t(g)$ is represented based on its local position according to the head root.

3.1.2 Virtual Character

The ways in which a virtual character are animated are based either on skeletal or blendshape-based key-frames. In both cases, a character's motion can be represented as $Y_{1:T} = [y_1, \ldots, y_T]$. As before, T denotes the total number of frames (both X and Y should have equal lengths; therefore, a simple time alignment process regularizes the corresponding data). Each y_t contains the pose of a character, which can also be defined as the rotation, q, of a joint angle, such as $y_t = [q_t(1), \ldots, q_t(K)]$, where $y_t \in \mathbb{R}^{dy}$ and K denotes the total number of joints of a virtual character. When the motion of a virtual character is represented by blendshapes, a frame of the animated sequences is defined as $y_t = [g_t(1), \ldots, g_t(F)]$, where $g_t(f)$ denotes the evolution (the weights) of the fth blendshape at the tth frame.

3.1.3 Time Alignment

In the MOP framework, the user is asked to perform actions that are later used to train the HMM. To properly provide the activity following functionality, X and Y should always have the same duration, T. Therefore, right after capturing the actions of a user for particular motions and before training the HMM, normalization of the corresponding durations is used to make sure both X and Y have the same lengths. Thus, a soft mapping between x_t and y_t is achieved.

3.2 HMM-Based Action Follower Mechanism

This section presents the HMM, which allows the user to control a virtual character based on its body activities. Two steps are presented. In the first step (training process), an HMM is trained to learn the parameters, λ, of the model. The parameters of the HMM are defined as $\lambda = \{a_{ij}, \pi_i, b_i\}$, where a_{ij} represents the state transition matrix and denotes the probability of making a horizontal transition from the ith to the jth state, π_i represents the prior vector and denotes the initial distribution vector over the sub-states of the model, and finally, b_i represents the observation probability distribution and indicates the probability of the production state. In the second step (decoding process), a user's activity is predicted in real-time. The HMM allows the user to animate a virtual character by controlling its motions independently based on different body activities. According to Berndt and Clifford [45], the developed model can be considered as a hybrid approach between HMM and DTW.

3.2.1 Training Process

The training process is based on setting an HMM to recognize the captured motion of the user. Since this approach does not consider capturing variations of a user's action, a single action is used to animate a single motion of a character. Thus, someone can easily realize that limited training data are available. A template-based DTW method is used, and a single example of the captured data is set for the learning process. The structure of the state of the HMM is set directly from a single template, which is a time sequence that is represented by $x_1, \ldots, x_T$ where x_t represents the posture of a user, as described in the previous section.

The $x_1, \ldots, x_T$ sequence is then used to create a left-to-right Markov chain that is represented by $S_1, \ldots, S_T$ states. Only two transitions are considered in this implementation: the self transition, a_{ii}, and the next transition, a_{ij}. The a_{ij} transition describes the state transition probability distribution between state i and $i+1$. In this case, it is also considered that the data are regularly sampled. Therefore, the transition probabilities are set manually as $a_{ii} = a_{ij} = 0.5$, where a value of 0.5 corresponds to an average transition time that is equivalent to the original sequence. Figure 2 illustrates the training process of the developed model and, more specifically, the ways the $S_1, \ldots, S_T$ states that represent the left-to-right Markov chain are constructed.

Considering an observation O (O is a measured vector of length T), the probability $b_j(O)$ in a state j is set to a Gaussian distribution with vector x_j as the center. Apparently, a single template is not enough to fully estimate the required distributions. Thus, a simplified form for estimating the required distributions is used based on the following equation:

$$b_j(O) \propto \exp\left[-\sum_{t=1}^{T} \frac{(O_t - x_{j_t})^2}{2 \times \sigma^2}\right] \tag{1}$$

where σ^2 denotes the standard deviation between the measured and template data.

3.2.2 Decoding Process

During the runtime of the application, the decoding algorithm runs on a growing sequence of observations, $O_{1:t} = O_1, \ldots, O_t$. The decoding process is responsible for estimating the probability distribution, $a_t(i)$, which denotes the probability of the observation sequence, $O_{1:t}$, as well as the state, S_i, at the tth time step of the given model. Then, it is possible to estimate $a_t(i)$ using the forward algorithm [46]. Three different quantities are computed from the distribution $a_t(i)$: the likelihood, L_t, at time t of the observation sequence, $O_{1:t}$ (see Eq. 2), the first moment, μ_t, of the normalized distribution, $a_t(i)/L_t$ (see Eq. 3), and the variance σ_t^2 of the normalized distribution (see Eq. 4) as presented below.

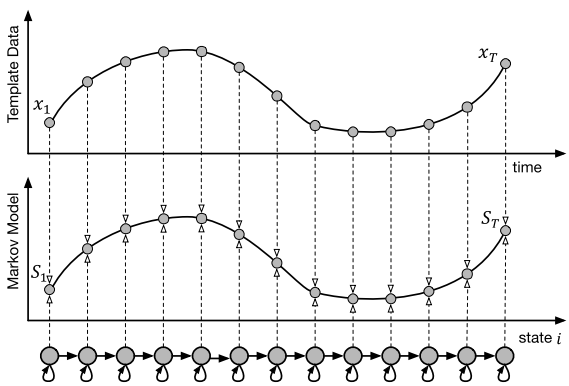

Fig. 2 A Markov chain $S_1, \ldots, S_T$ is generated based on the captured motion sequence $x_1, \ldots, x_T$ (template data) during the training process of the HMM

$$L_t = \sum_{i=1}^{N} a_t(i) \tag{2}$$

$$\mu_t = \sum_{i=1}^{N} \left[i \times \frac{a_t(i)}{L_t}\right] \tag{3}$$

$$\sigma_t^2 = \sum_{i=1}^{N} \left[(i - \mu_t)^2 \times \frac{a_t(i)}{L_t}\right] \tag{4}$$

The likelihood, L_t, is used to estimate the similarity between the observation and template sequences. The first moment, μ_t, is used to predict the time progression index of the input motion. This can be described as the essential output parameter of the real-time prediction process, since it enables the real-time alignment of the observation to the template sequence. This time progression is computed by μ_t/f, where f denotes the sampling frequency. Finally, the variance, σ_t^2, of the normalized distribution in the $a_t(i)$ state is important, since it can be used to invalidate some of the outputs from the activity follower methodology. It should be noted that the value of the variance could be considered complementary to the likelihood value, L_t. It provides information on the similarity between data (a small σ_t^2 value indicates that the progression index is statistically defined properly and vice versa).

To improve the efficiency of the real-time prediction process, the forward algorithm is calculated on a sliding window as:

$$a_1(i) = \pi_i \times b_i(O_1) \tag{5}$$

$$a_{t+1}(j) = k \times \left[\sum_{i=i_{inf}}^{i_{sup}} a_t(i) \times a_{ij}\right] \times b_j(O_{t+1}) \tag{6}$$

In Eq. 5, $i \in [1, N]$, and in Eq. 6, $i \in [1, T-1]$ and

$j \in [1, N]$. Moreover, π_i denotes the initial distribution in state i, a_{ij} denotes the transition probability distribution, and i_{inf} and i_{sup} denote the inferior and superior indexes of a sliding windows with lengths of, $2p$ (a 20 ms window is used in the examples presented in this paper). It should be noted that the MOP user can easily adjust the window size to make the recognition process work properly. In Eq. 6, k denotes a normalization factor that results from the truncation of the sum. Generally, this normalization can be ignored if p is a large number. Additionally, it should be noted that the computation of $a_{t+1}(j)$ is quite efficient, since the form of a_{ij} is rather simple. Finally, i_{inf} and i_{sup} are generally set as functions of the index μ_t based on the following rules:

$$\begin{aligned}
&\text{if} \quad (1 < \mu_t \leq p) \begin{cases} i_{inf} = 1 \\ i_{sup} = 2 \times p + 1 \end{cases} \\
&\text{if} \quad (p < \mu_t \leq N - p) \begin{cases} i_{inf} = \mu_t - p \\ i_{sup} = \mu_t + p \end{cases} \\
&\text{if} \quad (N - p < \mu_t \leq N) \begin{cases} i_{inf} = N - 2 \\ i_{sup} = N \end{cases}
\end{aligned} \tag{7}$$

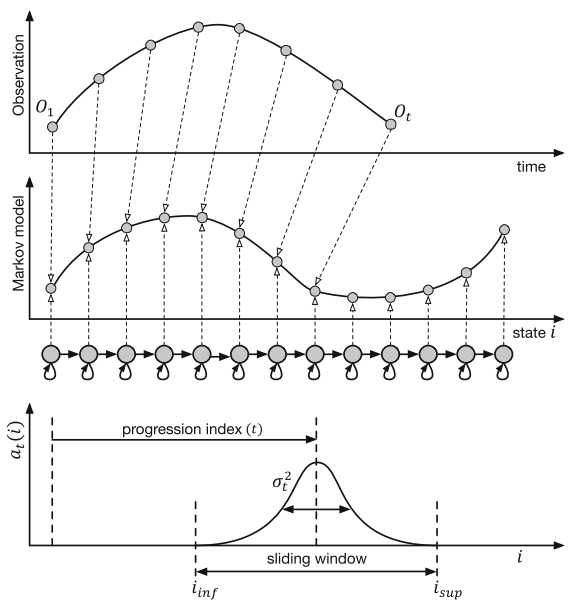

Fig. 3 A graphical representation of the decoding process as implemented in the MOP framework

Concluding, Fig. 3 illustrates the decoding process presented above and shows the way the observation data are aligned with the states of the Markov model (time warping) as well as the windowing process of the associated probability function.

4 Implementation Details and Evaluation

This section presents the interactive workflow of MOP as well as examples that illustrate its functionalities. MOP is also evaluated against similar frameworks/methods.

4.1 Implementation

MOP is implemented as an editor extension of the widely used game engine, the Unity3D.[1] Figure 4 illustrates the simple interface of MOP in Unity3D. The user defines the number of action-motion pairs, assigns the captured actions to the corresponding motions of the virtual character, and finally chooses the input sources that should be used. In the current version of the MOP framework, Microsoft's Kinect[2] motion capture sensor with the software development kit (SDK) provided by [47], and the Leap[3] motion controller with the SDK provided by [48] are integrated. A downloadable version of the Unity3D package that contains the complete framework with a variety of examples can be found on the project webpage of the MOP framework.

4.1.1 Interactive Workflow

A simple workflow (see Fig. 5) allows users to interact directly with the MOP framework. The typical workflow is split into three basic parts, as follows:

- *Action Capture* The user captures examples of actions that should be used to control the motions of the virtual character.
- *Training* The HMM, which is the core component of the MOP framework, is trained to recognize the user's actions.

[1] https://unity3d.com/.

[2] https://www.xbox.com/en-US/xbox-one/accessories/kinect.

[3] https://www.leapmotion.com/.

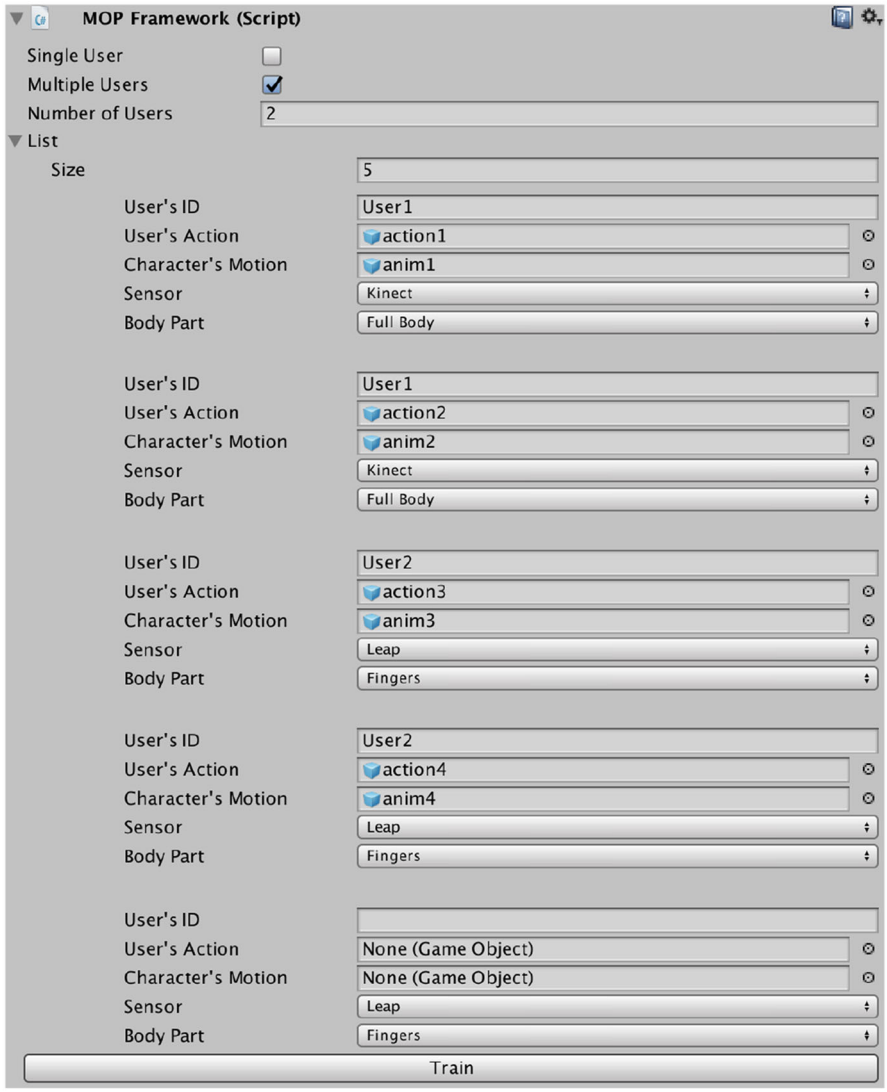

Fig. 4 The editor window in Unity3D that provides users the ability to author their own computer puppetry interaction scenarios

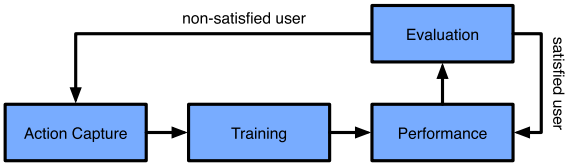

Fig. 5 The workflow of the presented method provides users the ability to directly and easily author the required computer puppetry scenarios through iterating the required steps

- *Performance* The user can use his/her body to animate the virtual character.
- *Evaluation* In this case, the user evaluates the results. Depending on the satisfaction of the user, a captured action might be replaced by a new action, or the user might need to adjust the window size that is used to recognize the input actions. This process continues until the user becomes satisfied with the displayed results based on his/her performance.

4.1.2 Examples

A variety of examples are given to illustrate the efficiency and flexibility of the MOP framework. The video that accompanies this paper illustrates all of the examples presented in this section.

Figure 6 illustrates a user controlling the expressions of a face model, the full-body motions of the

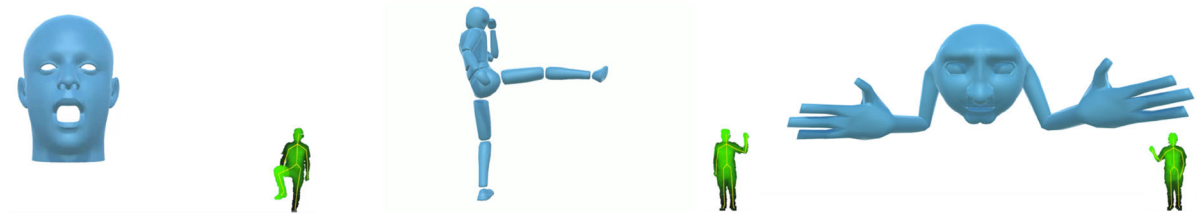

Fig. 6 A user controls a face model (right), a human-like character (middle), and a virtual creature (left) using his body

human character, and a virtual creature using his body. Figure 7 illustrates a single user controlling two characters using his body and also two users controlling a single character using their bodies. In the previously mentioned examples, the full-body motion of the user is captured using the Kinect motion capture sensor. Use of the Leap motion controller provides users the ability to control the virtual creature by using the fingers of both hands (see Fig. 8). MOP also allows users to assign multiple input sources to a single character. Given this functionality, a final example is presented in which the user controls the motion of a virtual creature using his face and fingers. In this example, the face of the user is captured using the Kinect motion capture device, and the user's fingers are captured using the Leap sensor. Figure 9 illustrates this example.

Here the following should be noted. When the user controls more than one character, the HMM model is trained independently for the user's action that should be assigned to each character's motion; thus, the user can control two characters simultaneously. When two users or more are responsible for the motion of a single character, the puppetry author should assign the actions of each user to the corresponding motions of the character. It should be noted that the actions of the users are captured independently although they are

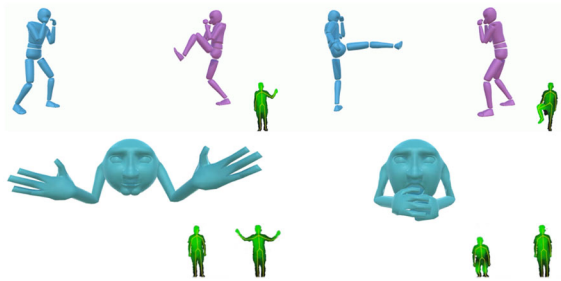

Fig. 7 A user controls two characters (top) using his body, and two users control a virtual creature (bottom) using their bodies

Fig. 8 A user controls a virtual creature using finger gestures from both hands

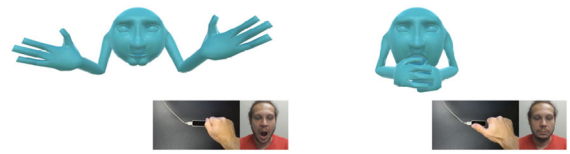

Fig. 9 A user controls a virtual creature using his face and fingers

treated as a combined entry. Finally, in cases where two users or more are responsible for controlling the motion of multiple characters, the actions of the users are captured independently, and they are treated independently as separate entries.

4.2 Evaluation

The following sub-sections evaluate MOP in terms of performance as well as against that of existing approaches.

4.2.1 Performance Evaluation

The evaluation process examines the performance of the MOP framework based on different character control scenarios. Table 1 presents the results of the evaluation process, which are based on an Intel i7-6700 CPU at 3.4 GHz with 16 GB of RAM and an NVIDIA GeForce GTX 1060 with 6 GB of RAM.

The presented methodology works in real time in all the examined character puppetry scenarios. Depending on the scenario, both the frames per second

Table 1 The performance of the MOP framework based on a variety of example scenarios

No. of motions	No. of characters	No. of input sources	No. of users	Sensor	Latency (ms)	FPS
Skeletal animation						
5	1	1	1	Kinect	7	89
10	1	1	1	Kinect	9	88
Non-skeletal animation						
5	1	1	1	Kinect	6	90
10	1	1	1	Leap	9	86
Hybrid animation						
5	1	1	1	Kinect	6	91
10	1	2	2	Kinect and leap	15	81
Multiple characters						
10	2	1	1	Kinect	11	85
15	3	2	2	Kinect and leap	17	79
Multiple users						
10	2	1	2	Kinect	11	83
10	2	2	2	Kinect and leap	16	81
Average					11.36	84.45

(FPS) and latency change. The results' values are all quite close to each other, and the performance of MOP remains on interactive framerates. The average framerate of the examined character control scenarios is approximately 84.45 FPS, and the latency is approximately 11.63 ms. The results suggest that this framework can be used in a variety of applications that are related to computer puppetry and motion control requiring run-time efficiency.

4.2.2 *Comparison with Previous Methods*

A comparison was conducted to illustrate the differences between the MOP framework and previous character animation approaches. Table 2 illustrates several criteria and functionalities that are divided into 12 categories, which were collected to compare MOP with other character animation frameworks and computer puppetry methods.

The first category, input data, examines the user's body part inputs, which can be used as control parameters to animate the virtual character. This category examines the possibility of using inputs retrieved from the full-body, fingers, face, and other sources. The second category, number of users, examines the possibility of the methods to provide functionality for multiple users. The third category, input sensors, examines the possibility of using multiple sensors; and the fourth category, input sources, examines the possibility of using single or multiple input sources (e.g., a combination of face and fingers). The fifth category, motion representation, examines the representation that a motion requires to animate a character. The sixth category, target character, examines the system's ability to handle human-like or non-human-like characters. The seventh category, number of characters, examines the possibility of animating multiple characters simultaneously. The eighth category, character animation, examines the possibility of synthesizing animation or simply displaying the animation of the virtual character. The ninth category, performance capture, examines how the user's motion is used to animate a character. Finally, the last three categories—system performance, naturalness, and dynamics—examine the possibility of the systems to work in real time, the naturalness of the final motion, and the use of physics-based animation, respectively.

The basic concept behind MOP is the need for easy-to-use, fast, flexible performance-driven computer puppetry authoring frameworks. By comparing MOP with the methods presented in Table 2, it can be easily stated that many functionalities can be integrated or considered to design a powerful computer puppetry

Table 2 Comparing previously developed computer puppetry approaches with the MOP framework

	Dontcheva et al. [33]	Ishigaki et al. [35]	Yamane et al. [36]	Seol et al. [34]	Rhodin et al. [8]	Chen et at al. [9]	MOP
Input Data							
Full-body		✓	✓	✓	✓	✓	✓
Fingers					✓		✓
Face					✓		✓
Other	✓						
Number of users							
Single user	✓	✓		✓	✓		✓
Multiple users						✓	✓
Input sensors							
Single sensor	✓	✓		✓		✓	✓
Multiple sensors					✓		✓
Input sources							
Single source	✓	✓		✓	✓		✓
Multiple sources						✓	✓
Motion representation							
Skeletal animation	✓	✓	✓	✓	✓		✓
Blendshapes					✓	✓	✓
Hybrid (skeletal and blendshapes)							✓
Target character							
Human-like characters	✓	✓			✓		✓
Non-human-like characters	✓		✓	✓	✓	✓	✓
Number of characters							
Single character	✓	✓	✓	✓	✓	✓	✓
Multiple characters							✓
Character animation							
Motion synthesis	✓	✓	✓	✓	✓	✓	
Motion display							✓
Performance capture							
Joint manipulation		✓					
Activity recognition	✓	✓		✓	✓	✓	
Activity follower							✓
System performance							
Real-time	✓	✓		✓	✓	✓	✓
Naturalness							
Natural-looking motion	✓	✓	✓	✓	✓	✓	✓
Dynamics							
Physics-based animation		✓					

authoring framework. Moreover, each of the existing studies has advantages and disadvantages. Finally, the comparison is based on the basic functionalities that each of the approaches provides. Even if the methods could be extended to use multiple sources or increase the number of users that could interact with the

content, the presented evaluation is based on what the authors presented and discussed in the corresponding papers.

The solutions that are closest to MOP in terms of translating the user's activity to that of the virtual character are those proposed by Rhodin et al. [8], Ishigaki et al. [35], Chen et at al. [9], and Seol et al. [34]. In terms of animating a variety of virtual characters (both humanoid and non-humanoid), the methods proposed by Rhodin et al. [8] and Seol et al. [34] can be characterized as similar to MOP. In regard to using multiple and different input sensors and sources, only the methods proposed by Rhodin et al. [8] can be considered similar to the MOP framework. In terms of the performance capture process, Ishigaki et al. [35] used activity recognition and direct joint manipulation to animate the virtual character. When examining the way that the motion of a character is displayed, it is obvious that only MOP allows the motion itself to evolve in time based on the activity-following method compared with the other approaches that use motion synthesis techniques.

Compared with Dontcheva et al. [33], a mapping process allows a user to animate a character in a natural-looking way. However, it can be stated that even if the character is able to follow the movements of the controller for each layer separately, the need for non-commercial specialized equipment for controlling a virtual character can be considered a disadvantage. The approach presented by Yamane et al. [36] provides the ability to animate a number of different non-human characters in a natural-looking way; however, the motion is synthesized off-line and does not provide the user the freedom to control the virtual character. It is also important to note that the system presented by Yamane et al. [36] is not a real-time performance-driven animation. Generally, the examined frameworks and methods cover only a limited number of functionalities and lack the generalization that is provided by the MOP framework. Moreover, none of the examined approaches provides the activity-following functionality for controlling the evolution of a virtual character's motion. Similarly, functionalities such as the control of hybrid motions (motions represented by skeletal keyframes and blendshape-based keyframes simultaneously), the simultaneous control of two characters, or the simultaneous use of multiple input sources are not provided by any of the examined approaches.

5 Conclusions and Future Work

This paper introduces an animation-by-demonstration computer puppetry authoring framework. The real-time character control is achieved using HMM to train and predict the evolution of the user's actions. By assigning the character controller mechanism to the forward algorithm, it is possible to predict the activity of the user that corresponds to the motion of the character that should be displayed as well as the evolution of the character's motion based on the evolution of the user's action. The development and demonstration of a variety of examples shows the flexibility of the presented methodology in handling diverse characters, inputs, and sources, thereby providing users the ability to control the required animation quite efficiently for various computer puppetry scenarios.

In its current form, the developed algorithm is able to handle a variety of different user actions. Since the recognition process takes into account the user's total number of captured joints, it is easy to predict the input action and display the necessary motion of the character for different body parts of the user. However, there are cases where the user might require similar actions to control the virtual character. An example includes actions where the user employs only his/her right hand. In this case, the system can recognize different actions; however, motions can be recognized incorrectly if they are too similar. We believe that such wrong estimations might affect the way the user controls the virtual character. Therefore, a future improvement of the MOP framework could provide feedback for users immediately after recording the action, indicating whether the new action would affect the recognition process. Additionally, we also believe that improving the recognition to generalize forms of user actions would also benefit the MOP framework, which means that users would be able to control the virtual character even if an action is performed with a variation of the captured one.

The character animation framework could be improved in additional ways besides the recognition process. For example, it would be interesting to integrate additional character control mechanisms that would enhance the flexibility and applicability of the current version. The addition of physics-related reactions [35, 49] could benefit the range of motion, which could be synthesized by the current versions. Other

functionalities that would improve the naturalness of the character's motion should also be considered. A simple example includes the relationship description [50], which would allow the developed framework to handle interactions between multiple characters more efficiently. Finally, the extension of the current HMM to a hierarchical model [51] with reactive interpolations [52] is another improvement that would benefit the MOP framework, since it would predict the progression of the character's motion while synthesizing the motion sequences, which is a functionality that is not provided in the current version of the MOP framework.

References

1. Sarris, N., & Strintzis, M. G. (2005). *3D modeling and animation: Synthesis and analysis techniques for the human body*. Hershey: IGI Global.
2. Gleicher, M. (1998). Retargetting motion to new characters. In *Annual conference on computer graphics and interactive techniques*, (pp. 33–42). New York: ACM.
3. McCann, J., & Pollard, N. (2007). Responsive characters from motion fragments. *ACM Transactions on Graphics*, *26*(3), 6.
4. Oshita, M. (2010). Generating animation from natural language texts and semantic analysis for motion search and scheduling. *The Visual Computer*, *26*(5), 339–352.
5. Mousas, C., & Anagnostopoulos, N. C. (2015). CHASE: Character animation scripting environment. In *ACM SIGGRAPH international conference on virtual reality continuum and its applications in industry*, (pp. 55–62).
6. Levine, S., Krähenbühl, P., Thrun, S., & Koltun, V. (2010). Gesture controllers. *ACM Transactions on Graphics*, *29*(4), 124.
7. Davis, J., Agrawala, M., Chuang, E., Popović, Z., & Salesin, D. D. (2003). A sketching interface for articulated figure animation. In *ACM SIGGRAPH/eurographics symposium on computer animation*, (pp. 320–328). Aire-la-Ville: Eurographics Association.
8. Rhodin, H., Tompkin, J., In Kim, K., Varanasi, K., Seidel, H. P., & Theobalt, C. (2014). Interactive motion mapping for real time charactercontrol. *Computer Graphics Forum*, *33*(2), 273–282.
9. Chen, J., Izadi, S., & Fitzgibbon, A. (2012). Kinêtre: Animating the world with the human body. In *ACM symposium on user interface software and technology*, (pp. 435–444). New York: ACM.
10. Ouzounis, C., Mousas, C., Anagnostopoulos, C.-N., & Newbury, P. (2015). Using personalized finger gestures for navigating virtual characters. In *Workshop on virtual reality interaction and physical simulation*, (pp. 5–14).
11. Lam, WC., Zou, F., & Komura, T. (2004). Motion editing with data glove. In *ACM SIGCHI international conference on advances in computer entertainment technology*, (pp. 337–342).
12. Wang, R. Y., & Popović, J. (2009). Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, *28*(3), 63.
13. Shiratori, T., & Hodgins, J. K. (2008). Accelerometer-based user interfaces for the control of a physically simulated character. *ACM Transactions on Graphics*, *27*(5), 123.
14. Slyper, R., & Hodgins, J. K.(2008). Action capture with accelerometers. In *ACM SIGGRAPH/eurographics symposium on computer animation*, (pp. 193–199). Aire-la-Ville: Eurographics Association.
15. Jacobson, A., Panozzo, D., Glauser, O., Pradalier, C., Hilliges, O., & Sorkine-Hornung, O. (2014). Tangible and modular input device for character articulation. *ACM Transactions on Graphics*, *33*(4), 82.
16. Mukai, T., & Kuriyama, S. (2005). Geostatistical motion interpolation. *ACM Transactions on Graphics*, *24*(3), 1062–1070.
17. Kovar, L., & Gleicher, M. (2003). Flexible automatic motion blending with registration curves. In *ACM SIGGRAPH/eurographics symposium on computer animation*, (pp. 214–224). Aire-la-Ville: Eurographics Association.
18. van Basten, B., & Egges, A. (2012). Motion transplantation techniques: A survey. *IEEE Computer Graphics and Applications*, *32*(3), 16–23.
19. Mousas, C., Newbury, P., & Anagnostopoulos, C.-N. (2013). Splicing of concurrent upper-body motion spaces with locomotion. *Procedia Computer Science*, *25*, 348–359.
20. Witkin, A., & Popovic, Z. (1995). Motion warping. In *Annual conference on computer graphics and interactive techniques*, (pp. 105–108). New York: ACM.
21. Glardon, P., Boulic, R., & Thalmann, D. (2004). PCA-based walking engine using motion capture data. In *IEEE computer graphics international*, (pp. 292–298).
22. Song, J., Choi, B., Seol, Y., & Noh, J. (2011). Characteristic facial retargeting. *Computer Animation and Virtual Worlds*, *22*(23), 187–194.
23. Ouzounis, C., Kilias, A., & Mousas, C. (2017). Kernel projection of latent structures regression for facial animation retargeting. In *EUROGRAPHICS workshop on virtual reality interaction and physical simulation*, (pp. 59–65).
24. Cashman, T. J., & Hormann, K. (2012). A continuous, editable representation for deforming mesh sequences with separate signals for time, pose and shape. *Computer Graphics Forum*, *31*(2), 735–744.
25. Levine, S., Wang, J. M., Haraux, A., Popović, Z., & Koltun, V. (2012). Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics*, *31*(4), 28.
26. Wei, X., Zhang, P., & Chai, J. (2012). Accurate realtime full-body motion capture using a single depth camera. *ACM Transactions on Graphics*, *31*(6), 188.
27. Raunhardt, D., & Boulic, R. (2011). Immersive singularity free full body interactions with reduced marker set. *Computer Animation and Virtual Worlds*, *22*(5), 407–419.
28. Liu, H., Wei, X., Chai, J., Ha, I., & Rhee, T. (2011). Realtime human motion control with a small number of inertial sensors. In *Symposium on interactive 3D graphics and games*, (pp. 133–140). New York: ACM.

29. Mousas, C., Newbury, P., & Anagnostopoulos, C. -N. (2014). Evaluating the covariance matrix constraints for data-driven statistical human motion reconstruction. In *Spring conference on computer graphics*, (pp. 99–106). New York: ACM.
30. Eom, H., Choi, D., & Noh, J. (2014). Data driven reconstruction of human locomotion using a single smartphone. *Computer Graphics Forum*, *33*(7), 11–19.
31. Min, J., & Chai, J. (2012). Motion graphs++: A compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics*, *31*(6), 153.
32. Sturman, D. J. (1998). Computer puppetry. *IEEE Computer Graphics and Applications*, *18*(1), 38–45.
33. Dontcheva, M., Yngve, G., & Popović, Z. (2003). Layered acting for character animation. *ACM Transactions on Graphic*, *22*(3), 409–416.
34. Seol, Y., O'Sullivan, C., & Lee, J. (2013). Creature features: online motion puppetry for non-human characters. In textitACM SIGGRAPH/eurographics symposium on computer animation, (pp. 213–221).
35. Ishigaki, S., White, T., Zordan, V. B., & Liu, C. K. (2009). Performance-based control interface for character animation. *ACM Transactions on Graphics*, *28*(3), 61.
36. Yamane, K., Ariki, Y., & Hodgins J. (2010). Animating non-humanoid characters with human motion data. In *ACM SIGGRAPH/eurographics symposium on computer animation*, (pp. 169–178). Aire-la-Ville: Eurographics Association.
37. Mousas, C., & Anagnostopoulos, C. N. (2017). Performance-driven hybrid full-body character control for navigation and interaction in virtual environments. *3D Research*, *8*(2), 18.
38. Otsuka, T., Nakadai, K., Ogata, T., & Okuno, H. G. (2011). Incremental bayesian audio-to-score alignment with flexible harmonic structure models. In *ISMIR*, (pp. 525–530).
39. Françoise, J., Caramiaux, B., & Bevilacqua, F. (2012). A hierarchical approach for the design of gesture-to-sound mappings. In *Sound and music computing conference*, (pp. 233–240).
40. Bettens, F., & Todoroff, T. (2009). Real-time dtw-based gesture recognition external object for max/msp and puredata. In *Sound and music computing conference*, (pp. 30–35).
41. Bevilacqua, F., Zamborlin, B., Sypniewski, A., Schnell N., Guédy, F., & Rasamimanana, N. (2010). Continuous realtime gesture following and recognition. In *Gesture in embodied communication and human-computer interaction*, (pp. 73–84). Berlin: Springer.
42. Fels, S. S., & Hinton, G. E. (1993). Glove-talk: A neural network interface between a data-glove and a speech synthesizer. *IEEE Transactions on Neural Networks*, *4*(1), 2–8.
43. Fiebrink, R., Cook, P R., & Trueman, D. (2011). Human model evaluation in interactive supervised learning. In *SIGCHI conference on human factors in computing systems*, (pp. 147–156). New York: ACM.
44. Mori, A., Uchida S., Kurazume, R., Taniguchi, R. I., Hasegawa, T., & Sakoe, H. (2006). Early recognition and prediction of gestures. In *IEEE international conference on pattern recognition*, (pp. 560–563).
45. Berndt, D. J., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *KDD workshop*, (pp. 359–370).
46. Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.
47. Microsoft. Kinect SDK from https://www.microsoft.com/en-us/kinectforwindows/, (2017).
48. Leap Motion. Developers SDK from https://developer.leapmotion.com/, (2017).
49. Liang, X., Hoyet, L., Geng, W., & Multon, F. (2010). Responsive action generation by physically-based motion retrieval and adaptation. In *Motion in games*, (pp. 313–324). Berlin: Springer.
50. Al-Asqhar, R. A., Komura, T., & Choi, M. G. (2013). Relationship descriptors for interactive motion adaptation. In *ACM SIGGRAPH/eurographics symposium on computer animation*, (pp. 45–53).
51. Mousas, C., & Anagnostopoulos, C.-N. (2017). Real-time performance-driven finger motion synthesis. *Computers & Graphics*, *65*, 1–11.
52. Mousas, C. (2017). Full-body locomotion reconstruction of virtual characters using a single inertial measurement unit. *Sensors*, *17*(11), 2589.