Pattern Recognition 62 (2017) 73–86

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Approximate string matching: A lightweight approach to recognize gestures with Kinect

Rodrigo Ibañez, Álvaro Soria *, Alfredo Teyseyre, Guillermo Rodríguez, Marcelo Campo

Department of Software Engineer, ISISTAN Research Institute (CONICET-UNCPBA), Campus Universitario, Paraje Arroyo Seco, Tandil, Buenos Aires, Argentina

ARTICLE INFO

Article history:
Received 4 June 2015
Received in revised form
14 July 2016
Accepted 20 August 2016
Available online 23 August 2016

Keywords:
Natural user interfaces
Gesture recognition
Machine learning
Kinect
Approximate string matching

ABSTRACT

Innovative technologies, such as 3D depth cameras, promote the development of natural interaction applications in many domains among large audiences. In this context, supervised machine learning techniques have been proved to be a flexible and robust approach to perform high level gesture recognition from 3D joints provided by these depth cameras. This paper proposes a lightweight approach to recognize gestures with Kinect by utilizing approximate string matching. The proposed approach encodes the movements of the joints as sequences of characters in order to simplify the gesture recognition as a widely studied string matching problem. We evaluated our approach by applying other widespread used techniques in the research field. The experimental evaluations show that the proposed approach can obtain relatively high performance in comparison with the state-of-the-art machine learning techniques. These findings provide further evidence that our approach could be a viable strategy for recognizing gestures, even in devices with medium and low processing capability (e.g., smartphones, tablets, etc.).

1. Introduction

In recent years, depth sensors have become increasingly popular, thus reducing not only their cost but also their size. Among the most popular devices that allow for creating a virtual representation of the objects captured in the scene are Kinect,[1] ASUS Xtion,[2] SoftKinetic,[3] Structure Sensor,[4] Leap[5] and Meta.[6] These devices can be purchased at an average cost of 150 USD, in comparison with similar previous devices, which cost between 35,000 and 500,000 USD [1].

Particularly, Kinect identifies people and obtains the position of 20 human body parts in the 3D space in real time. Developers of Natural User Interface (NUI) applications can exploit this feature by generating a 3D representation of the human skeleton that mimics the movements of a person and even recognizes his/her gestures.

In this sense, several full-body gesture recognition approaches have emerged to facilitate the human–computer interaction. Initially, rule-based approaches relied on a set of parameters and thresholds on body part locations to recognize static postures and simple movements of the body parts [2–4]. For example, rule-based approaches allowed for determining whether the right hand was above the head or whether the left hand moved to the right in a certain timespan. The weakness of these approaches is that each gesture must be defined manually, thus requiring a considerable effort to both define the rules for gestures that involve complex movements of the body parts, and then, test the correctness of the defined rules. Furthermore, manual definition of gestures is a daunting and error-prone process that requires users with vast domain knowledge and experience.

To address these issues, other approaches have explored supervised machine learning techniques for gesture recognition [5,6]. These techniques require a set of labeled training gestures to learn and subsequently identify a new given gesture as one of the learned gestures. For example, Bhattacharya et al. used Support Vector Machines (SVM) and Decision Trees (DT) for gesture recognition in a military application [7]. Another successful approach for gesture recognition is based on the Dynamic Time Warping algorithm (DTW) [8]. However, these studies have scarcely presented any comparative performance analysis of utilized algorithms.

In this context, the novelty of this paper stems from the combination of well-known approximate string matching algorithms

* Corresponding author.
E-mail addresses: rodrigo.ibanez@isistan.unicen.edu.ar (R. Ibañez), alvaro.soria@isistan.unicen.edu.ar (Á. Soria), alfredo.teyseyre@isistan.unicen.edu.ar (A. Teyseyre), guillermo.rodriguez@isistan.unicen.edu.ar (G. Rodríguez), marcelo.campo@isistan.unicen.edu.ar (M. Campo).

[1] http://www.microsoft.com/en-us/kinectforwindows/
[2] http://www.asus.com/Multimedia/Xtion_PRO/
[3] http://www.softkinetic.com/
[4] http://structure.io/
[5] https://www.leapmotion.com/
[6] https://www.spaceglasses.com/

http://dx.doi.org/10.1016/j.patcog.2016.08.022
0031-3203/

Table 1
Characterization of gesture recognition research works.

Approach	Machine learning technique	Assessment	Recognition method
Hong et al. (2000)	K-Means	Homemade dataset and skin color tracker	Statistical
Stiefmeier et al. (2007)	HMM, DTW	Homemade dataset, Pentium 4 (3 GHz, 1 GByte RAM) and Matlab	Statistical
Fothergill et al. (2012)	DT	MSRC-12 Kinect gesture dataset	Learning
Waithayanon et al. (2011)	DTW	Homemade dataset, MS Kinect, Kinect for Windows SDK Beta	Learning
Bhattacharya et al. (2012)	SVM, DT	Military air force dataset and LIBSVM library	Learning
Ibañez et al. (2014)	DTW, HMM	Homemade dataset and Kinect SDK	Learning
Jiang et al. (2015)	WDTW, SPW	ChaLearn Gesture dataset, Matlab 7.12.0, Dell PC with Duo CPU E8400	Statistical (PCA), one-shot learning
Hachaj et al. (2015)	HMM,	Homemade dataset, C# with HMM libraries from Accord Framework	Statistical (PCA)
Slama et al. (2015)	KM, TSVM, TWG, LTBSVM	MSR-action 3D, UT-Kinect and UCF-Kinect datasets. PC Intel Core i5-3350P (3.1 GHz) CPU, 4 GB RAM a PrimeSense camera	Learning, Statistical (Grasmann manifold)
Gu at al. (2012)	HMM	Homemade dataset, MS Kinect, OpenNI and NITE	Statistical
Celebi et al. (2012)	DTW	Homemade dataset and Kinect SDK	Learning

[9] with gesture recognition with Kinect, as a lightweight approach that allow devices with different processing capabilities to recognize gestures. The proposed approach encodes the movements of the joints as sequences of characters in order to simplify the gesture recognition as a string matching problem. We compared the accuracy and performance of our gesture recognition approach with other well-known techniques used in NUI applications, such as Dynamic Time Warping (DTW) [10,11], Procrustes Analysis [12], Markov Chains [13], and Hidden Markov Models [14,15]. Moreover, we analyzed our approach by exchanging approximate string matching for a string matching algorithm [16]. To conduct the experiments, we used the public Microsoft Research Cambridge (MSRC-12) Kinect gesture dataset which involves 30 people performing 12 different gestures. These gestures come from a first person shooter game and a music player [17] and are detailed in Section 4. The experimental evaluations show that the proposed approach achieves higher performance rates (in terms of time of processing and CPU) than the state-of-the-art algorithms, even than the one based on string matching. These promising results, in terms of accuracy and performance, indicate that devices with low or medium processing capability would recognize gestures.

The remainder of this paper is organized as follows: Section 2 reports an overview of related works. Section 3 presents an in-depth description of the proposed approach. Section 4 discusses the experiments and results along with the lessons learned. Finally, Section 5 presents the conclusions and identifies future lines of work.

2. Related work

The literature includes several approaches for human gesture recognition by capturing body movement on by video – for a review of the state of the art in human movement recognition see [18–22,7,23,8,24]. Despite these meaningful research efforts, accurate recognition of human body movements was found to be significantly difficult and challenging [25].

The emergence of new cost-effective depth cameras, such as Kinect, has promoted the development of natural interaction applications in many domains, and particularly, opened up new opportunities to improve human activity and gesture recognition [26]. By using Kinect, data for computing recognition can be obtained directly from the Red–Green–Blue (RGB) camera coupled with a depth sensor or from the Kinect SDK's skeleton tracking API. On the one hand, several research works have utilized raw data from the RGB camera and depth sensor to track human silhouettes by considering temporal continuity constraints of human motion information and computing centroids for each activity, based on contour generation [27–30]. On the other hand, other attempts successfully adapted machine learning techniques to gesture recognition using Kinect's skeleton data [7,17,8]. The Microsoft Kinect SDK's skeleton-tracking API allows efficient and real-time body tracking, which is instrumental in the development of recognition tools [31]. Although Kinect is able to estimate the position of various body parts, it still demands considerable effort from developers to add hoc recognize gestures.

To cope with this issue, our research addresses gesture recognition using the skeleton-tracking API by exploring supervised machine learning, which provides a flexible and robust alternative by considering gesture recognition as a classification problem [19]. In this context, a classification problem consists in labeling a gesture consistently. For example, some of the machine learning techniques that have been also widely applied in gesture recognition are Support Vector Machine (SVM), Decision Tree (DT), Dynamic Time Warping (DTW) and Hidden Markov Models (HMM), among others. Table 1 shows the criteria to characterize reviewed research works that explore skeleton-tracking API for gesture recognition. We have classified the reviewed works into the following categories: machine learning technique, assessment, development environment and recognition method. Machine learning technique describes algorithms used in the approaches, such HMM, DTW, and SVM, among others. Assessment lists datasets used in the experimentation, along with programming languages, libraries, operating systems, external devices or characteristics of the computers used in the experiments. Recognition method presents methods used to recognize gestures, such as statistical, graphical or learning.

In line with our research, Gu et al. described the implementation of a non-intrusive, real-time gesture recognition system using a depth sensor. The authors explained how to obtain body-joint features from the skeleton model generated by the Kinect sensor, and supported the modeling of gestures by utilizing HMMs [32]. Another approach successfully applied an algorithm based on Dynamic Time Warping (DTW) [8]. Although this method was successfully instantiated for certain applications with high classification accuracy, developers had to implement complex algorithms and also perform the training process ad hoc. In particular, Fothergill et al. addressed the problem of collecting gesture datasets to improve the accuracy and performance of the system based on machine learning algorithms [17]. Celebi et al. proposed a weighted DTW method that weights joints by optimizing a discriminant ratio. They also demonstrated that the recognition of the weighted DTW outperfomed the conventional DTW [33]. Along this line, Jiang et al. recommended a novel multi-layered gesture recognition method with Kinect; this approach implements a reconstruction-based gesture recognition method based on principal component analysis (PCA), and utilizes Weighted Dynamic Time

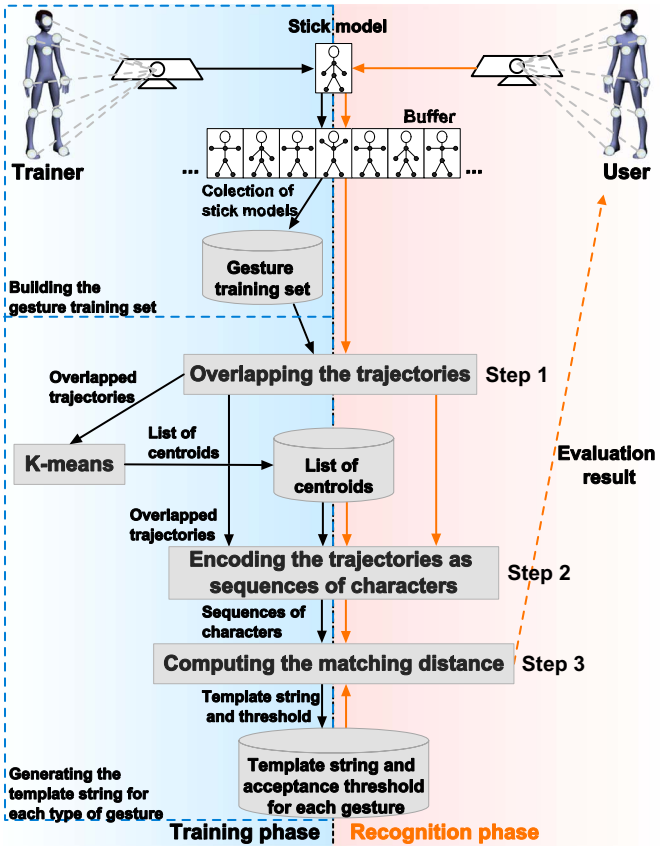


Fig. 1. Approximate string-matching approach based on a supervised machine-learning process.

Warping (WDTW) for the location of classified components [34]. Hachaj and Ogiela proposed human-actions recognition schema using angle-based and state-of-the-art coordinates-based features and multivariate continuous HMM classifier with Gaussian distribution based on PCA. The problem addressed in this research has been the classification of gym warm-up and fitness exercises [35]. In a previous work, EasyGR (Easy Gesture Recognition), a tool based on DTW and HMM, which helps to reduce the effort involved in gesture recognition development, was presented [36]. EasyGR user interface allows non-specialists to record, edit, and store gestures, enabling them to easily create a new training set. The main difference between the studies described above and EasyGR is that we have addressed the experimental evaluation of the approach, not only in terms of accuracy of the recognition techniques, but also in terms of the development effort in practice.

SVM has also been successfully applied in several works. Bhattacharya et al. applied SVM and DT to recognize aircraft gestures used in the military Air Force [7]. Slama et al. introduced a representation of a skeleton-joint motion in a compact and efficient way that leads to an accurate action recognition. The approach was evaluated on public datasets by using simple Karcher Mean (KM), one Tangent SVM (TSVM), Truncated Wrapped Gaussian (TWG) and Local Tangent Bundle SVM (LTBSVM). The approach combines linear dynamic modeling with statistical analysis on a manifold (Grassmann manifold), avoiding the boundary and the monotonicity constraints presented by the classical DTW algorithm [37]. Unlike contemporary research on this topic, our approach aims to provide developers with a development framework to recognize gestures regardless of the domain.

String matching algorithms have already been used for gesture recognition. For example, Hong et al. recognized 2D gestures by modeling each gesture as a Finite State Machine and applying a variant of Knuth–Morris–Pratt algorithm [38]; however, this algorithm was reported much lower than DTW and HMM. Stiefmeier et al. reported on the use of a set of sensors mounted on the lower limbs, the upper limbs, and the torso of the body to calculate the relative positions of the hands in the 3D space [39]. For each two consecutive positions of the hand, they compute a vector whose direction belongs to one of the eight quadrants of the Cartesian space. As each quadrant is labeled with a letter from *a* to *h*, after a period of time the 3D positions are encoded as a sequence of characters, which enables applying string matching. Although the authors apply approximate string matching to recognize gestures, their work differs from ours in that they use a set of intrusive sensors instead of Kinect. In particular, action recognition methods designed for motion capture with sensors might not be suitable for depth cameras because of the difference in the motion data quality resulting from noise and occlusions [40]. In this context, we claim that the application of string matching algorithms to gesture recognition from Kinect's body-joints requires further research.

3. Our approximate string matching approach to recognize gestures

The Kinect SDK identifies the position of 20 body joints in a 3D space (x, y, z), thus generating a virtual representation of the human body called “stick model”. These positions are updated 30 times per second, so when we observe the stick model for a period of time, we obtain the movements of the joints that represent gestures.

In this work we propose to encode the movements of the joints as sequences of characters in order to simplify the gesture recognition as a string-matching problem [41]. String matching problems are widely studied in computer science, and basically consist in searching a sequence of characters called “string” in a text, i.e., another sequence of character. In this sense, when encoding gestures as sequences of characters, recognizing a gesture means finding a sequence in the text.

In order to encode gestures as sequences of characters and use string matching as a gesture recognition approach, we follow the classical machine learning process, which consists of two phases: training phase and recognition phase. Fig. 1 shows a conceptual view of our string matching approach for recognizing gestures with Kinect. The figure also depicts the training and the recognition phases along with their steps.

The training phase involves building a gesture training set, and generating a template string for each type of gesture. The training set is a list of sample gestures for each type of gesture, i.e., it contains not only instances of different gestures but also different instances of the same gesture. To build this set, a trainer stands in front of Kinect and performs the movements corresponding to the first training gesture. As the trainer moves, Kinect tracks the 3D positions of his/her body joints and generates a stick model 30 times per second. Tracking the trainer's body joints for a period of time results in a collection of stick models, which represents the intended gesture. Finally, the collection of stick models is stored persistently in the gesture training set. This process of performing a gesture and storing the corresponding collection of stick models is repeated several times with each of the different gestures to be trained.

Once the gesture training set is populated, the approach is ready to perform the second part of the training phase: generation of the template strings. A template string is a particular sequence of characters that identifies the correct way to perform each gesture. To obtain these template strings, the approach is fed with the stored collections of stick models, and a number of steps take

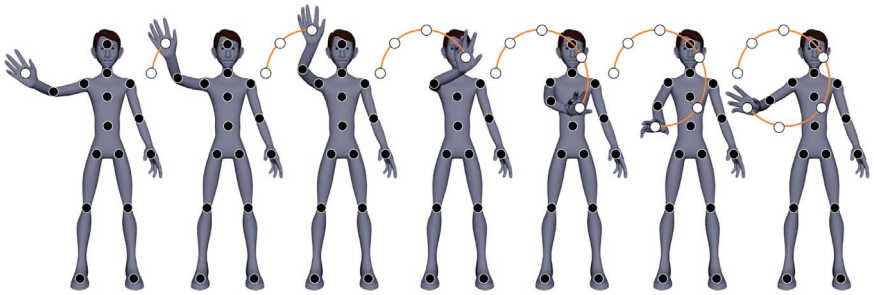


Fig. 2. Sequence of seven stick models performing the *Circle* gesture.

place: overlapping the trajectories, encoding the trajectories as sequences of characters, and computing the matching distance.

The first step refines the gesture training set in order to improve the recognition accuracy, which may be reduced when people have different body builds and/or perform gestures in different locations within the Kinect detection field. Hence, to execute the overlapping step and make trajectories invariant to these issues, the approach considers the collection of stick models as a collection of trajectories, each of which corresponds to the movement of one body joint. Then, the step consists in overlapping the trajectories that correspond to the same body joint in different performances of the same type of gesture. This process involves aligning trajectories, normalizing distances, and fixing or removing any inaccurate 3D joint positions generated by Kinect.

After the trajectories have been overlapped, the second step consists in encoding all of them as sequences of characters to provide a suitable input for applying string matching. To encode the trajectories, the approach firstly applies the *k*-means algorithm [42], which groups all the 3D points of the trajectories that belong to the same 3D region in *k* clusters and generates a list of *k* numbered centroids. Then, we encode each point of the trajectory with the number of the centroid that is nearest it, thus turning the trajectories into sequences of characters.

Finally, the third step consists in computing the matching distances among all the sequences of characters that represent the same gesture. To measure the matching distance between each pair of sequences, approximate string matching is utilized. Approximate string matching computes these differences or distances among the sequences in terms of the minimum number of operations needed to match the sequences [9,43]. These operations, also known as edit operations, involve the *insertion* of a character in the first sequence, the *deletion* of a character from the first sequence, and the *replacement* of a character of the first sequence with the corresponding character of the second sequence. Once we quantify the differences among sequences, we are able to assess whether two sequences are more similar to each other than a third sequence. Thus, after applying approximate string matching between each pair of sequences, we select as template string the sequence that is the most similar to the remaining sequences. Furthermore, the longest distance as acceptance threshold for the gesture is selected. This threshold allows for determining if a new sequence of characters corresponding to a gesture and a template string are similar enough for the gesture to be recognized.

As a result of the training phase, we have the template string and the acceptance threshold for each of the trained gestures, thus enabling the approach to start with the second phase. The recognition phase involves feeding our approach with an unknown gesture so that the approach be able to identify the type of the gesture. This phase starts when a user stands in front of Kinect and performs a gesture. Then, as in the training phase, the steps of overlapping the trajectories and encoding them as sequences of characters are performed. In the step of encoding the trajectories as sequences of characters, the recognition phase differs from the training phase in that, instead of applying the *k*-means algorithm, we assign one character to each of the 3D positions by using the generated list of centroids in the training phase. Furthermore, both phases also presents a difference in the step of computing the matching distance. This difference is that, rather than using the sequences of characters to train the string matching technique, the recognition phase involves comparing the sequences with the template strings, i.e., the technique measures the distance between the new sequences and each of the template strings. Then, if the distance is lower than one of the acceptance thresholds, the gesture is recognized and the evaluation result is presented to the user. On the other hand, if the distance is higher than all the acceptance thresholds, we select the nearest threshold as the intended gesture, thus leaving the user the decision of accepting the gesture as valid.

In order to understand how to recognize gestures by applying string matching, we present the following example. It consists in developing a hand gesture recognizer whose main goal is to recognize the figures that the user draws in the air. For simplicity, we will focus on recognizing a *Circle* performed with the right hand. Fig. 2 shows a sequence of seven stick models of a person performing the *Circle* gesture.

The black and white dots represent the body joints contained in each stick model. The white dot indicates the position of the right hand in each stick model, and the orange line indicates the progression of this position in the performance of the gesture. At the end of the sequence, we can observe the whole progression of the right hand position, called trajectory. We denote the trajectory of the right hand as $T_{r_{hand}} = (x_1, y_1, z_1)\ldots(x_7, y_7, z_7)$, where x_i represents the horizontal coordinate, y_i represents the vertical coordinate, and z_i represents the depth coordinate, all of them with regard to the Kinect position, which represents the coordinates of the origin $(0, 0, 0)$. Note that each of the body joints of the stick model generates a trajectory in the same way as shown in Fig. 2.

To recognize the *Circle* gesture, our approach requires the trainer to generate several samples of the gesture. Each sample is a collection of stick models, which is represented by the trajectories of the joints; the approach considers each collection of stick models as a collection of trajectories, one for each joint, and constitutes the training set for the *Circle* gesture. To facilitate the understanding of the approach, let us assume that the trajectory corresponding to the right hand is enough for the technique to recognize the *Circle* gesture. It is worth noting that it may be necessary to compare the trajectories corresponding to several joints to recognize some gestures.

Once the training set has been built, the approach is ready to learn the particularities of the movements that represents the *Circle* gesture. This is the starting point for the training phase. Here, the idea is to encode all trajectories in the training set as sequences of characters, and select one of these sequences as a template string. The template string represents all the sequences

that describe the intended gestures, and is used as a model to recognize new gesture during the recognition phase.

The following sub-sections present an in-depth description of the steps of our approach to obtain the template string by differentiating the training and recognition phase. Section 3.1 describes the overlapping of trajectories during the training phase, where the trajectories are made suitable for comparison; Section 3.2 presents the encoding of the trajectories as sequences of characters along the training phase; Section 3.3 introduces the computation of the matching distance also in the training phase; finally, Section 3.4 describes the same steps during the recognition phase.

3.1. Step 1: Overlapping the trajectories

The process of overlapping the trajectories involves three tasks: (i) making the trajectories suitable for comparison, (ii) subtracting the *Centroid* from each point of the trajectory, and (iii) dividing all the points of each trajectory by a scale factor. As for the first task, the 3D position of the gesture trajectories can drastically vary since the trainers and the users may be in different locations within the Kinect detection field and may also have different body builds. Hence, we need to transform the trajectories to make them invariant to different positions of the trainer and body builds.

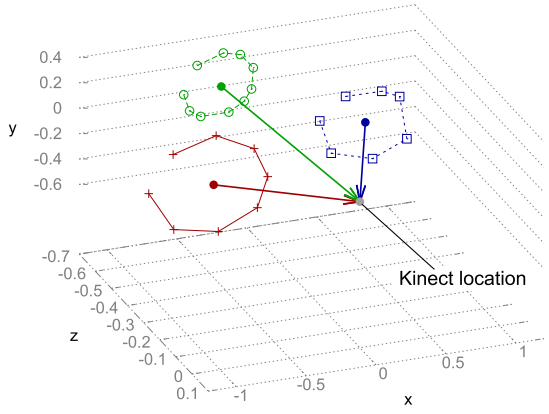
The transformation starts by moving each trajectory to one common point, naturally the origin $(0, 0, 0)$. Fig. 3a shows three trajectories of the right hand in different performances of the *Circle* and illustrates how they are moved to the origin. Then, the *Centroid* of each trajectory using Eq. (1) is calculated. The *Centroid* is a 3D point that represents the geometric center of the trajectory, and it is calculated by adding all the points of the trajectory and dividing the result by the number of points n:

$$Centroid = (\bar{x}, \bar{y}, \bar{z}) = \frac{\sum_{i=1}^{n}(x_i, y_i, z_i)}{n} \quad (1)$$

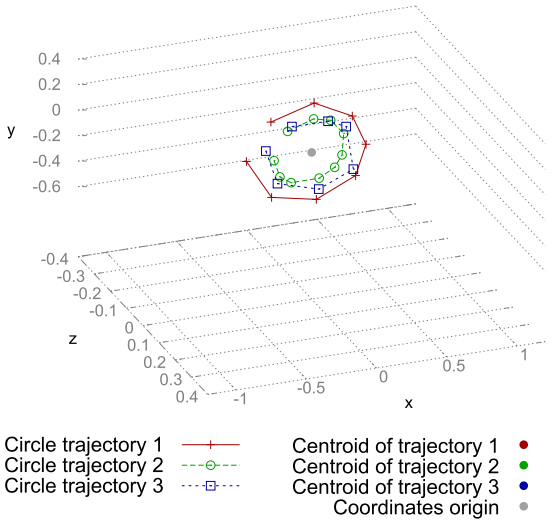
$$(x_i, y_i, z_i)' = (x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z}) \quad (2)$$

The second task consists in subtracting the *Centroid* from each point of the trajectory using Eq. (2), so that the new *Centroid* of the trajectory becomes the origin. As a result, the trajectories are centered and invariant to the trainer's position (Fig. 3b).

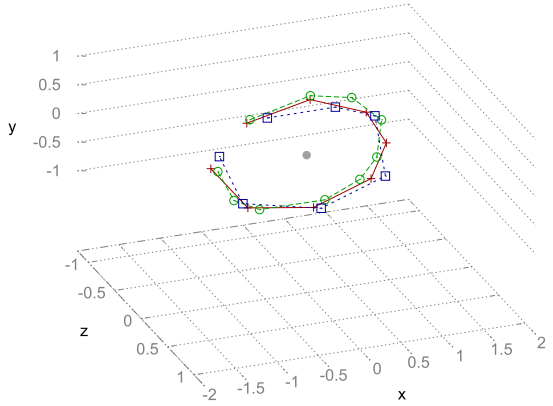
At this point, the centered trajectories need to be normalized to make them invariant to the trainer's body build. Furthermore, not only the size of the trajectories produced by people with different body builds but also the trajectories produced by the same person in different performances of the *Circle* might be different. Thus, the transformation process continues by calculating a scale s factor of



(a) Trajectories of the *Circle* gesture performed in different locations



(b) Trajectories of the *Circle* gesture after being centered



(c) Trajectories of the *Circle* gesture after being centered and normalized

Fig. 3. Process of centering and normalizing trajectories.

each trajectory, which is a statistical measure of the size of the trajectory, by using the following equation:

$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i, y_i, z_i)'^2}{n}} \tag{3}$$

The third task consists in dividing all the points of each trajectory by the scale factor using Eq. (4), so that the new scale factor of each trajectory becomes 1. Consequently, all the joint positions have magnitudes ranging from 0 to 1, and all the trajectories perfectly fit in a cube of size 2, from $(-1, -1, -1)$ to $(1, 1, 1)$, centered in the origin (Fig. 3c).

$$(x_i, y_i, z_i)'' = \left(\frac{x_i}{s}, \frac{y_i}{s}, \frac{z_i}{s}\right) \tag{4}$$

At this point, the trajectories have been centered and normalized, so that the trajectories corresponding to the same gestures can be visually aligned, i.e., the points of the trajectories are proximal. However, to make a suitable string matching comparison, the continuous values of the trajectories need to be encoded as a sequence of characters. This means that applying string matching over the continuous value of the trajectories could cause a mismatch because the 3D points of each trajectory are not the same as those in Fig. 3, i.e., the points are proximal but are not exactly the same. Therefore, all the points of the trajectories are transformed into discrete values before applying string matching.

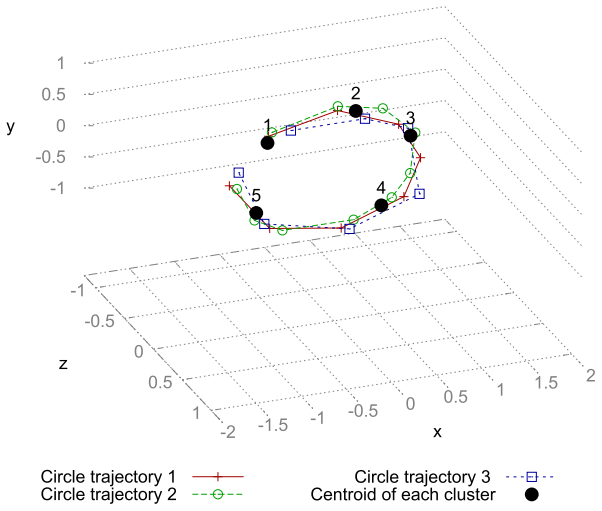
3.2. Step 2: Encoding the trajectories as sequences of characters

To encode the 3D point of the trajectories from continuous into discrete values, our approach groups the 3D points that are proximal and assigns one character to them. Encoding each point as a character forces the trajectories to contain finite types of characters, thus enabling trajectories to be compared using string matching. Firstly, our approach applies the *k*-means algorithm to group the points of the trajectories that belong to the same region in *k* clusters. To group these points, *k*-means randomly assigns all the points of the trajectories to one of the *k* clusters and calculates the *Centroid* of each cluster by using Eq. (1). Then, *k*-means evaluates all the points and reassigns each point to the cluster that contains the nearest *Centroid* according to the proximity criterion (Eq. (5)). This criterion is the Euclidean distance between the point (P_i in the equation) and the cluster *Centroid* (C_j in the equation). After moving all the points to the corresponding cluster, the algorithm recalculates the *Centroid* of each cluster. This process is repeated until the convergence of the algorithm is reached, i.e., until all the points are in the correct group and no point movement is required. Eq. (5) calculates the Euclidean distance between the point P_i and the *Centroid* C_j:

$$D(P_i, C_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \tag{5}$$

Secondly, our approach assigns one character to each point of the trajectories, thus turning the trajectories into sequences of characters. As each cluster has a numbered *Centroid*, each point of the trajectories is conveniently encoded as the identification number of the nearest *Centroid*. That is to say, each point is encoded as the number of the cluster to which the point belongs. In this way, the trajectory is turned into a numeric sequence equivalent to the original. Fig. 4 shows the *Centroids* of the clusters obtained with its identification number after applying *k*-means with five clusters over the three trajectories of the *Circle*. Furthermore, Fig. 4 shows the result of encoding the trajectories as sequences of characters.

As trajectories may contain different numbers of points, when the number of points of one trajectory is higher than the number



$Trajectory\,1\,(S_1)$:	1	2	3	3	4	4	5	5		
$Trajectory\,2\,(S_2)$:	1	2	2	3	4	4	4	5	5	5
$Trajectory\,3\,(S_3)$:	1	2	3	4	4	5	1			

Fig. 4. Trajectories encoded as sequences of characters by using the identification number of the nearest centroid.

of clusters, two or more points of the trajectory are encoded with the same character. In this context, applying string matching among sequences of different lengths could cause a mismatch. Hence, we remove the contiguous repeated characters from each sequence before computing the matching distance between them. Thus, after removing the contiguous repeated characters from the sequences in Fig. 4, two of the three different trajectories generate exactly the same sequence [1, 2, 3, 4, 5].

3.3. Step 3: Computing the matching distance

Once all the trajectories of the training set have been encoded as sequences of characters, the approach selects a template string and defines the corresponding acceptance threshold. The template string is the sequence of characters in the training set that most closely resembles the remaining sequences; moreover, this template string is used as a model to recognize subsequent sequences, i.e., the subsequent sequences will be compared with the template string. The acceptance threshold represents the matching distance between the sequences in the training set that differ the most. That is, a new sequence of characters is recognized as an intended gesture if the distance between this sequence and the template string is shorter than the acceptance threshold. Therefore, the acceptance threshold is a value between 0 and the longest distance between all the sequences of characters in the training set that represents the same gesture. It is important to note that there is one template string and one acceptance threshold for each type of gesture.

To obtain the template string, our approach calculates the matching distance between each pair of sequences. The matching distance between two sequences of characters is computed with approximate string matching [43]. The goal of this technique is to transform one sequence into the other sequence by applying three types of operations: insertion, deletion, and replacement of characters. Then, the distance between the sequences can be measured as the smallest number of operations required for matching the sequences. The simplest way to implement approximate string matching is to penalize each applied operation with distance 1, and when no operation is required with distance 0. Another way is to penalize with different distances by applying a function of the involved character. Particularly, our approach penalizes each

operation by using the Euclidean distances between the centroids that represent each of the characters concerned.

Algorithm 1 shows the pseudocode for computing the distance between two sequences of characters by applying approximate string matching. The inputs of the algorithm are two sequences of characters and the list of *Centroids*, whereas the output of the algorithm is the distance between the sequences. To compute the distance between the sequences, the algorithm uses a local matrix to iteratively accumulate the minimum distance to transform each i character of the sequence S into each j character of the sequence T (lines 5–10). This distance is computed by adding the result of the *DistanceBetweenCharacters* function and the minimum distance among three previously computed distances, each of which corresponds to an insertion $D_{i-1,j}$, a deletion $D_{i,j-1}$, and a match/replacement $D_{i-1,j-1}$ (line 8). The *DistanceBetweenCharacters* function computes the distance between two single characters as the Euclidean distance between the *Centroids* that identify these characters.

Algorithm 1. Approximate string matching using dynamic programming.

Inputs
1: Input S, sequence of character of size n.
2: Input T, sequence of character of size m.
3: Input C, list of centroids of size k.
Outputs
1: Output distance between the sequences.
Local
1: Local D, matrix of size $n * m$.
Steps
1: **function** StringMatchingDistance (S, T, K)
2: $D_{i,0} \leftarrow \infty$
3: $D_{0,j} \leftarrow \infty$
4: $D_{0,0} \leftarrow 0$
5: **for** $i = 1 \rightarrow |S|$ **do**
6: **for** $j = 1 \rightarrow |T|$ **do**
7: $d \leftarrow$ DistanceBetweenCharacters (S_i, T_j, C)
8:

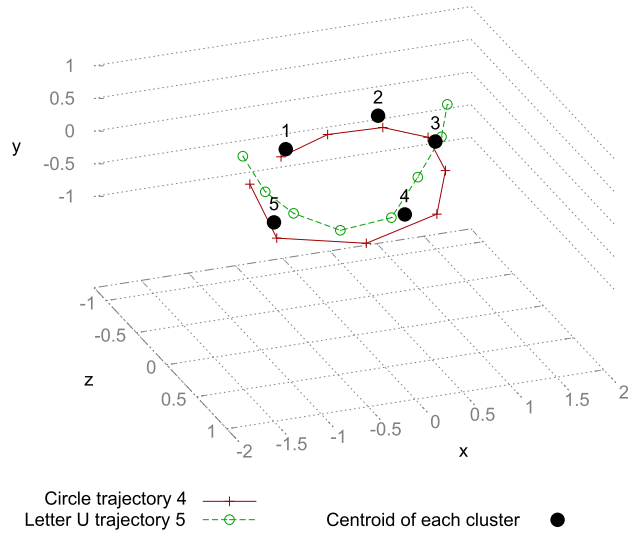
$$D_{i,j} \leftarrow d + \min\begin{cases} D_{i-1,j} & \leftarrow \textit{Insertion} \\ D_{i,j-1} & \leftarrow \textit{Deletion} \\ D_{i-1,j-1} & \leftarrow \textit{Match/Replacement} \end{cases}$$

9: **end for**
10: **end for**
11: **return** $D_{|S|-1,|T|-1}$
12: **end function**

1: **function** DistanceBetweenCharacters (s, t, C)
2: $u \leftarrow C_s$
3: $v \leftarrow C_t$
4: $\Delta \leftarrow (v_x - u_x)^2 + (v_y - u_y)^2 + (v_z - v_z)^2$
5: **return** $\sqrt{\Delta}$
6: **end function**

Table 2
Matching distance between each pair of sequences.

Distances	S_1	S_2	S_3	Total distance
S_1	0	0	1.113	1.113
S_2	0	0	1.113	1.113
S_3	1.113	1.113	0	2.226



$Trajectory\,4\,(S_4)$:	1	1	2	3	3	4	4	5	1
$Trajectory\,5\,(S_5)$:	1	5	5	4	4	3	3	3	

Fig. 5. Trajectories encoded as sequences of characters by using the identification number of the nearest *Centroid*.

Following the example of the *Circle* gesture, Table 2 shows the matching distance between each pair of sequences, and the addition of the distances from each sequence to the others, i.e., the total distance. As explained above, S_1 and S_2 are exactly the same sequence; thus the distances between them is 0. Furthermore, the distance between each of these sequences and S_3 is also the same distance 1.113. As the sequence S_1, corresponding to *Trajectory* 1, obtained the lowest total distance, this sequence is selected as the template string for the *Circle* gesture. Finally, the longest distance between the sequences that differ the most is selected to define the upper limit of the acceptance threshold (1.113 as shown in Table 2).

3.4. Recognition phase

So far, by considering our motivating example, we have obtained the template string and the acceptance threshold for the *Circle*, thus enabling the approach to start the recognition phase and recognize new *Circle* gestures. During the recognition phase, when a new gesture is performed by a user, the sequence of characters that corresponds to the right hand is compared with the template string of the *Circle* gesture, thus generating a distance value. If this value falls inside the acceptance threshold, from 0 to 1.113, then the gesture is recognized.

Let us suppose that the user performs two movements representing a *Circle* gesture and a letter *U* gesture, with his/her right hand, and the approach has to recognize the valid one, i.e., the *Circle* gesture. Fig. 5 shows the *Circle* trajectory (solid line) and the *U* trajectory (dashed line) after being centered and normalized. The black points represents the *Centroids* of each cluster and the above numbers identifies each *Centroid* (step 1: Overlapping the trajectories). Furthermore, Fig. 5 shows how the trajectories are encoded as sequence of characters, by using the number of the *Centroid* to which each point is nearest (step 2: Encoding the trajectories as sequences of characters). The *Centroid* numbers are the same as those generated in the training phase (Section 3.2). To recognize the gestures, we compute the matching distance between each sequence and the template string of the *Circle* by using approximate string matching (step 3: Computing the matching

Table 3
Gestures used in the experiment from MSRC-12 dataset.

Gesture group	Gesture ID	Gesture name	Gesture description
Music player	G1	Start music/raise volume	Raise outstretched arms
Music player	G5	Wind up the music	Make circular movements clockwise with right hand and counter-clockwise with left hand
Music player	G7	Take a bow to end the session	Bend forward at the waist, once and again
Music player	G9	Protest the music	Pause and rest your hands on your head
Music player	G10	Navigate to next menu	Slide right hand, palm down in front of you, from left to right
Music player	G11	Lay down the tempo of a song	Beat the air with both your hands
First person shooter game	G2	Crouch or hide	Squat down or crouch
First person shooter game	G3	Change weapon reach over your left shoulder with your right hand and then	Reach over your left shoulder with the right hand and bring both hands in front of you
First person shooter game	G4	Put on night vision goggles to change the game mode	Bring your hands up to your eyes
First person shooter game	G6	Shoot with a pistol	Stretching your arms out in front of you to form a pistol and make a recoil
First person shooter game	G8	Throw an object such as a grenade	With the right arm, make an overarm throwing movement
First person shooter game	G12	Kick to attack an enemy	Karate kick forwards with your right leg

distance). Then, as is expected, S_4 is recognized as a *Circle* with distance 1.113, which is shorter than to the acceptance threshold, and the S_5 is not recognized with distance 4.771, which is longer than the acceptance threshold.

Overall, we have proposed the use of approximate string matching as a viable gesture recognition technique. Although we exemplify the approach with a one-joint example for simplicity, gestures that involve movements of more than one body joint can be recognized following the same steps as those explained above. Furthermore, we have applied *k*-means with five clusters but other numbers of clusters could be better in different situations. Varying the number of clusters *k* increases or decreases the number of points of the template string, thus directly influencing the accuracy of the gesture recognition. A lower *k* value makes the recognition technique more tolerant, whereas a higher *k* value makes it less tolerant, i.e., the sequence of characters that corresponds to the imitated trajectory must be closer to the template string to be recognized. These and other variations are evaluated in the next section.

4. Experimental results

This section describes the experiments to assess the accuracy and performance of different gesture recognition techniques in comparison with our approach. To test the gesture recognition techniques, we utilized the MSRC-12 dataset [17], which involves 30 people performing 12 different gestures (a total of 6244 gesture instances), which are depicted in Table 3. The table shows the gesture ID, the gesture group (i.e., music player or first person shooter game), the gesture name and the gesture description. The demographics of the participants were 5′0″–6′6″ tall with an average of 5′8″, and aged 22–65 with an average of 31 years of age. In particular, this dataset is a list of files, each of which containing the track of 20 body joints of a person during a period of time; that is to say, each file is a collection of stick models generated by Kinect. As each file contains more than one instance of each gesture, we had to split the file to carry out the experiment. To perform the splitting, we used the EasyGR tool [36] that easily allowed us to individualize each gesture instance and check its correctness.

Once the dataset was split and its correctness was checked, we applied a 30-fold cross-validation strategy [44] to estimate the accuracy of the different techniques. In order to build a more realistic experiment, we tested the cross-subject accuracy of the techniques by partitioning the samples into 30 groups, each of which contained the gestures performed by each person. We used 29 groups for training and one group for testing the technique. We repeated the process 30 times with each of the 30 groups used exactly once as the validation data for testing. For each iteration, we not only counted the number of correctly classified gestures, but also built a confusion matrix that helped us to detect possible mislabeled classification of gestures and performed a more detailed analysis than mere proportion of correct guesses. Finally, all the results from the folds were averaged to produce a single estimation.

To evaluate accuracy, we measured the percentage of gesture correctly recognized by each technique; whereas to evaluate performance, we measured the spent time and consumed memory by the different techniques during the previous experiment. The following sub-sections describe the gesture recognition techniques, and the metrics and results after running the aforementioned techniques. Section 4.1 introduces the gesture recognition techniques used in the experiments along with a variant of our approach that applies string matching instead of approximate string matching. Section 4.2 describes a method to address the *k* initialization problem in *k*-means. Section 4.3 reports on the results from the viewpoint of accuracy, whereas Section 4.4 reports on the results from the performance viewpoint. Finally, lessons learned are stated in Section 4.5.

4.1. Gesture recognition techniques

Particularly, we compared our approach against Dynamic Time Warping (DTW) [10], Procrustes Analysis [12], Markov Chains [13], Hidden Markov Models [14], and a variant of our approach that uses simply string matching [16]. The selection of these techniques stems from the fact that are ones of the most used machine learning techniques in the contemporary research works reviewed in Section 2. *DTW* algorithm is originally used to find the similarity between two time series, i.e., sequences of values measured over a time interval. The algorithm optimally aligns the series by iteratively stretching and shrinking the time axis so as to minimize the distance between the series. These series represent the trajectories of the right hand in our example, and as a result of applying DTW, we obtain a distance value that measures the similarity between the trajectories. Thus, when training DTW with a set of trajectories of the right hand, we obtain an acceptance threshold just as when training approximate string matching.

Procrustes Analysis finds the optimum alignment between two shapes, modeled as a finite number of points, by applying a series of mathematical transformations that modify the shapes. In our context, these shapes represent the *Circle* trajectories, i.e., we see the whole trajectory as a static shape. The Procrustes Analysis involves three specific transformations: centering, normalizing,

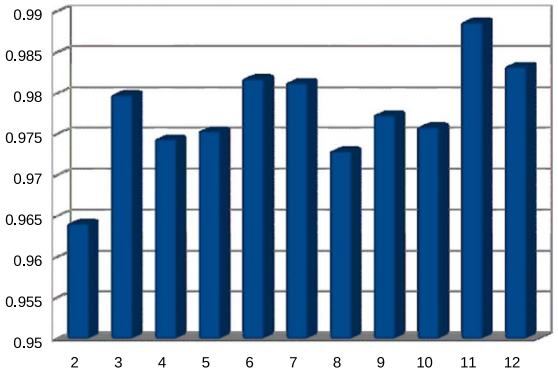
and rotating the trajectory. The first two transformations are the same as those applied in the step 1 (i.e., overlapping the trajectories), whereas the last transformation entails rotating the trajectories until the minimum distance among them is obtained. The minimum distance criterion is the addition of the Euclidean distances between each pair of points of the trajectory. The result of applying Procrustes Analysis is a distance value likewise DTW.

The two variants of *Markov Models* require the trajectories to be encoded as finite state sequences, thus we encode the trajectories as described in Section 3.2. Once the trajectories are expressed as sequences of states, they are used to feed each of the variants. One of the variants is *Markov Chains*, and it is used to represent certain stochastic processes characterized by being stateless. These processes are used to model the behavior of one or more variables as a function of the time, with the particularity that the values of the variables are independent of their previous values. Visually a Markov Chain is a finite state machine with probabilistic state transitions. In our context, these probabilities are computed by using the training sequences of the *Circle*. Then, when a new gesture is performed, the goal is to determine if the sequence corresponding to the right hand trajectory is a valid state transition. The other variant is the *Hidden Markov Models*, and similar to Markov Chains, it is used to represent stateless stochastic processes, but with unknown parameters, i.e., hidden states. Visually, a Hidden Markov Model is a probabilistic finite state machine, which is applied to solve three canonical problems: (1) given a trained model, i.e., the states and the probabilities among them, find the probability of generating a specific sequence; (2) given a trained model, find the sequence of hidden states that generated an specific sequence; (3) given a set of sequences, find the transition probabilities among states. In our example, solution (3) is used to train the models, and when a new gesture is performed, the sequence corresponding to the right hand trajectory is evaluated with solution (1).

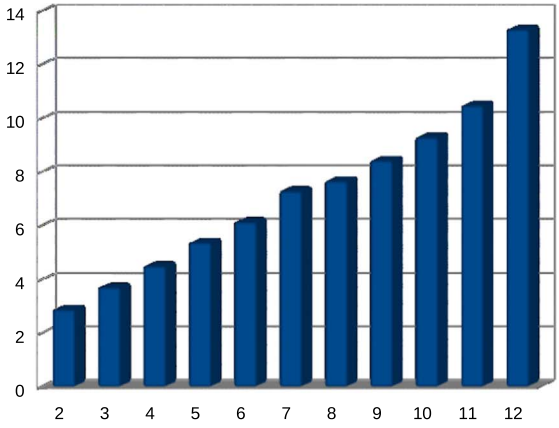
Finally, we have implemented a variant of our approach that just uses simply *string matching* instead of *approximate string matching*. This variant encodes the trajectories as sequences of characters and uses a set of accepted sequences for each gesture. Then, when a new gesture is performed, the corresponding sequence of characters is searched in all the sets by using exact matching.

4.2. Initialization of k

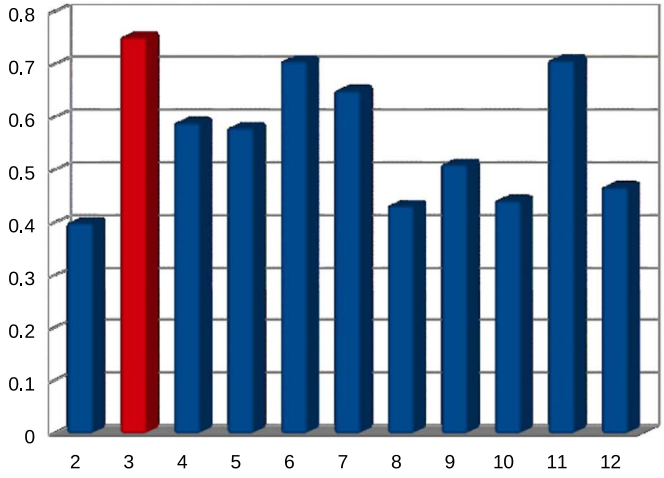
Most of the aforementioned machine learning techniques requires the initialization of k to perform the k-means algorithm. To tackle the initialization problem of k-means, we propose to utilize the weighted sum model (WSM) [45]. WSM is the simplest and most widely used multi-criteria decision making method for evaluating a number of alternatives simultaneously in terms of



(a) Accuracy



(b) Evaluation Time (ms)



(c) Weight Sum of accuracy (0.6) and evaluation time (0.4). Optimal cluster: 3

Fig. 6. Accuracy, evaluation time and weighted sum to determine number of clusters of ASM.

some decision criteria (e.g., accuracy, training time, evaluation time, training memory, and evaluation memory). In particular, the variables are: *accuracy*, *tt = training time*, *tm = trining memory*, *et = evaluation time*, and *em = evaluation memory*. Since the range of values of the variables has widely varied, a normalization was applied to re-scale values to $[0, 1]$, where 0 means the lowest performance and 1 the highest performance. Note that all variables, except for accuracy, were re-scaled using the following equation:

$$x' = 1 - \frac{x - \min(x)}{\max(x) - \min(x)} \tag{6}$$

The method assumes that for each criterion C_j there is a weight w_j, which denotes the importance of the criterion j, and there is a value a_{ij}, which denotes the performance of the alternative i when it is evaluated in terms of the criterion j. The method also assumes that all the values are normalized; therefore, the best alternative is the one with the largest cumulative value Eq. (7). Note that the sum of the weights of all criteria for each synthetic indicator must be 1, which represent 100% of the importance of the synthetic indicator:

$$A_i^{WSM-score} = \sum_{j=1}^{n} w_{j*} a_{ij}, \quad \text{for } i = 1, 2, 3, \ldots, m \tag{7}$$

In this way, we created 4 different synthetic indicators by establishing different weights in order to meet specific needs.

- $A1 = \alpha_{1*}accuracy + \beta_{1*}evaluation\ time$
- $A2 = \alpha_{2*}accuracy + \beta_{2*}evaluation\ time + \gamma_{2*}$
  $evaluation\ memory + \delta_{2*}training\ time + \omega_{2*}training\ memory$
- $A3 = \alpha_{3*}accuracy + \beta_{3*}training\ time + \gamma_{3*}training\ memory$
- $A4 = \alpha_{4*}accuracy + \beta_{4*}evaluation\ time + \gamma_{4*}evaluation\ memory$

We performed an experiment varying the number of clusters and selected the best for each technique, i.e., the number of clusters that allow each technique to obtain the best performance based on WSM. By taking into account the indicator $A1$, we aimed to find a balance between accuracy (α_1= 0.6) and evaluation time (β_1= 0.4). For example, Fig. 6 shows the accuracy, evaluation time and their weighted sum for approximate string matching (Y-axis respectively) while varying the number of clusters from 2 to 12 (X-axis). Particularly, we selected 3 clusters for approximate string matching (ASM-3), 6 for string matching (String-6), 3 for Markov Chains (Markov-3), and 5 for Hidden Markov Models (HMM-5). Fig. 6c shows the selection of k for ASM, and the selected value was 3 because it is the best balance between accuracy and evaluation time. Moreover, remaining indicators (A2, A3 and A4) are used in Section 4.5 to compare the techniques used in the experiments.

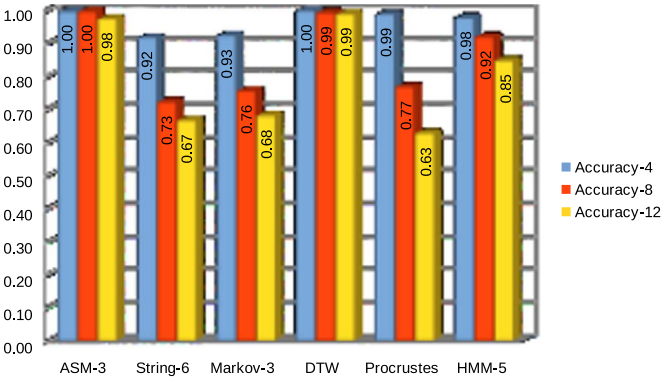


Fig. 7. Graphic that compares the number of gesture instances correctly classified for each technique depending on the number of different types of gestures used for training.

4.3. Accuracy of the approximate string matching technique

To determine to what extent the number of different types of gestures used for training impacted on the accuracy of the techniques, we varied the number of concurrently recognized gestures among 4 gestures (G1, G2, G3, G4), 8 gestures (G1, G2, G3, G4, G5, G6, G6, G7, G8), and 12 gestures (G1, G2, G3, G4, G5, G6, G6, G7, G8, G9, G10, G11, G12). By utilizing the proposed method mentioned above, we have obtained the k values required by most of the used machine learning techniques.

Fig. 7 illustrates the experimental results for ASM-3, String-6, Markov-3, DTW, Procrustes, and HMM-5. The techniques are listed along the horizontal axis, whereas the accuracy obtained by each technique is listed along the vertical axis. Each of the techniques has 3 bars that represents the accuracy obtained when the number of concurrently recognized gestures changes (first bar represents the technique accuracy for 4 concurrently recognized gestures, second bar represents the technique accuracy for 8 concurrently recognized gestures, and third bar represents the technique accuracy for 12 concurrently recognized gestures). HMM-5 lacks of an accuracy value for recognizing 12 gestures concurrently due to the unacceptable time spent in the cross-validation experiment (we stopped the experiment after several weeks running without obtaining any result). From the chart, we can observe that as the number of concurrently recognized gestures increases, the techniques' accuracy for almost all techniques decreases. ASM-3 and DTW are the most resistant to this change; for example, ASM-3 decreases the accuracy from 1 to 0.98 when the number of gestures increases from 8 to 12, whereas DTW decreases the accuracy from 1 to 0.99 when the number of gestures increases from 4 to 8 and also preserves this accuracy when the number of gestures increases from 8 to 12. On the other hand, Procrustes is the least resistant, decreasing the accuracy from 0.99 to 0.63 when the number of gestures concurrently recognized increases from 4 to 12.

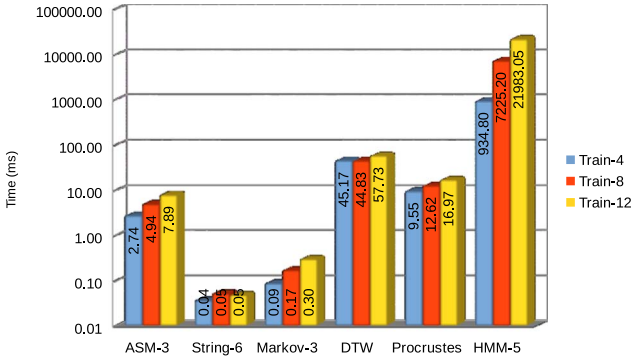
To visualize a more detailed analysis of our approach, we built a confusion matrix (Table 4) of ASM-3 when recognizing 12 different gestures concurrently. The confusion matrix shows how the predictions are made by the technique. In particular, the rows indicate the known type of gesture and the columns indicate the predictions made by the classifier. The value of each element in the matrix is the number of predictions made divided by the total number of tested gestures for each type, thus, the sum of all the cells of each row must be 100%. All correct predictions are located along the diagonal of the table, and the off-diagonal elements show the errors made by the techniques. Note that the average of the values of the diagonal represents the accuracy of the technique; for most of the gestures, ASM-3 works satisfactorily. The classification errors occur if two gestures are considerably similar to each other. For example, the technique confuses G4 with G6 since both movements involve the arms; G4 involves covering the face with the arms, and G6 involves performing a shooting movement, in which the user raises the arms until cover his face. Another example of confusing similar gestures is G7 with G2; G7 involves performing a reverence flexing the legs, and G2 involves performing a squat. The techniques also confuse G1 with G9; G1 involves raising the arms above the head, and G9 involves raising the arms and touching the head. Despite this confusion, ASM-3 still shows an acceptable accuracy, achieving 98% correct recognition rates.

4.4. Performance of the approximate string matching technique

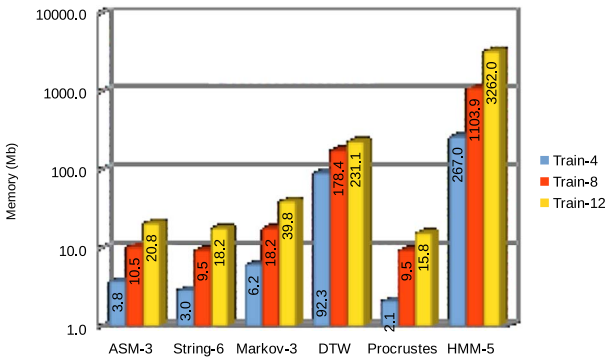
To evaluate the performance, we measured training time, evaluation time, training memory, and evaluation memory by the different techniques during the previous experiment in the

Table 4
Confusion matrix of approximate string matching for MSRC-12 dataset (accuracy: 0.976).

Known \ Predicted	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	94.7%	0.0%	0.0%	0.0%	0.0%	0.6%	0.0%	0.0%	4.7%	0.0%	0.0%	0.0%
G2	0.0%	99.4%	0.0%	0.0%	0.0%	0.0%	0.6%	0.0%	0.0%	0.0%	0.0%	0.0%
G3	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
G4	0.0%	0.0%	0.0%	94.1%	0.0%	5.9%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
G5	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
G6	0.0%	0.0%	0.0%	1.2%	0.0%	98.2%	0.0%	0.0%	0.0%	0.6%	0.0%	0.0%
G7	0.0%	5.3%	2.4%	0.0%	0.0%	0.0%	92.4%	0.0%	0.0%	0.0%	0.0%	0.0%
G8	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%
G9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%
G10	0.0%	0.0%	0.0%	0.0%	0.0%	1.2%	0.0%	0.0%	0.0%	98.8%	0.0%	0.0%
G11	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%
G12	0.0%	0.0%	1.2%	0.0%	0.0%	0.6%	0.0%	2.9%	0.0%	1.8%	0.0%	93.5%

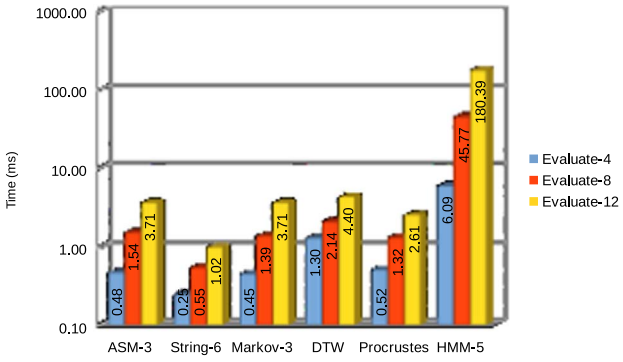


(a) Time to train each technique while varying the number of gestures (logarithmic scale)

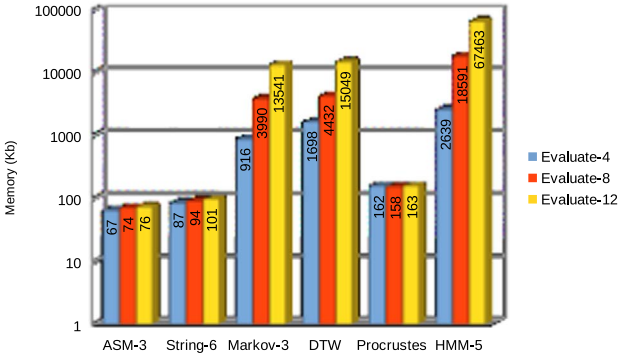


(b) Average memory footprint allocated by the train process to train each technique while varying the number of gestures (logarithmic scale)

Fig. 8. Training performance.



(a) Time to evaluate one sample (logarithmic scale)



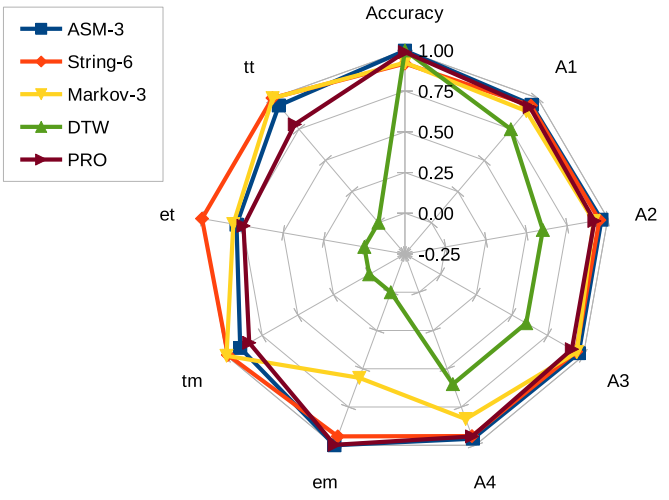
(b) Average memory footprint allocated by the recognition process to evaluate all the samples belonging to one fold (logarithmic scale)

Fig. 9. Evaluation performance.

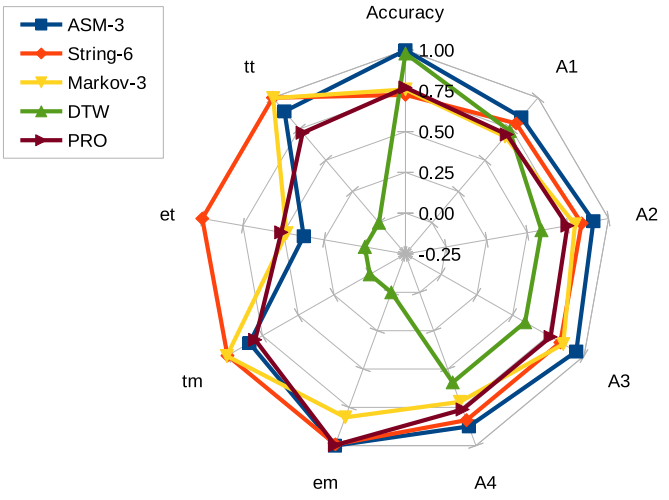
training phase and recognition phase. To measure the performance we used the native API of C#. Specifically, to measure the times we used the *stopwatch class*[7] that allows us to accurately measure elapsed time. To measure the consumed memory we used a method[8] of the *process class* that allows us to obtain the number of bytes currently allocated in memory by the associated process, also known as memory footprint. The experiment was run on a computer with an Intel(R) Core(TM) i7-3632QM CPU @ 2.20 GHz with 6 GB of RAM in a single thread. Note that we have measured the performance of each technique 30 times (one per fold).

Figs. 8a and b show the average time needed to train the techniques, and the average max number of bytes allocated in memory by the train process, respectively. Horizontal axis lists the different techniques, each of which having 3 different bars

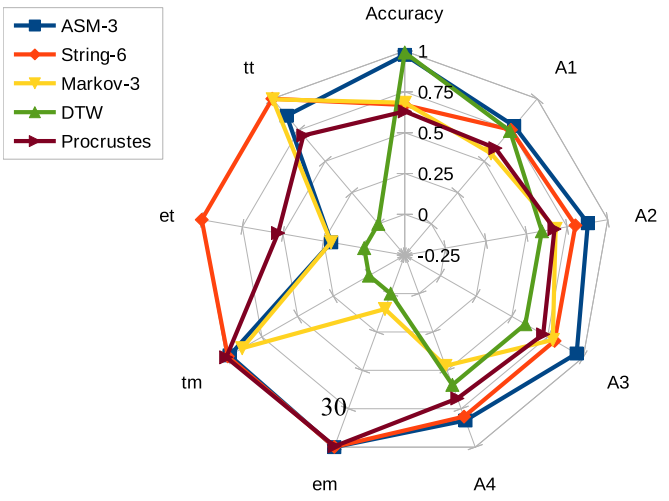
[7] https://msdn.microsoft.com/library/system.diagnostics.stopwatch
[8] https://msdn.microsoft.com/en-us/library/system.diagnostics.process



(a) 4 gestures



(b) 8 gestures



(c) 12 gestures

Fig. 10. Radar charts for 4, 8 and 12 gestures being concurrently recognized.

corresponding to the number of gestures used for training (first bars correspond to 4 gestures, second bars correspond to 8 gestures, and third bars correspond to 12 gestures). From the charts, we can observe that both graphs present a similar tendency; when the number of trained gestures increases, the performance for all techniques decreases. That is to say, the low performance is related with the increment in the time needed to train the techniques and the memory footprint allocated by the training process. String-6 and Markov-3 were clearly the best techniques regarding time consumption, followed by ASM-3, Procrustes, DTW, and finally HMM-5. On the other hand, all techniques except for HMM-5 and DTW required a similar amount of memory to be trained.

Likewise, Figs. 9a and b show to the average time needed to evaluate one sample and the average max number of bytes allocated in memory by the recognition process while evaluating all the samples belonging to one fold, respectively. These two figures are broadly similar to the previous one by preserving the same ranking. From Fig. 9a it is also noticeable that all the techniques, except for HMM-5, recognize a sample in less than 4.4 ms. This is relevant, since we must ensure a smooth interaction for natural user interfaces, i.e., a real-time gesture recognition. Fig. 9b shows that ASM-3, String-6, and Procrustes required a similar amount of memory. ASM-3 was the best technique requiring 25% less memory than the remaining techniques. Particularly, ASM-3 required 5% of the total memory required by DTW, which was the most accurate technique. On the other hand, HMM-5 was the worst technique requiring at least 22% more memory than the remaining techniques. Note that memory usage may became a critical factor in limited resource devices (e.g., smartphones).

4.5. *Lessons learned and threats to validity*

We have compared the proposed approach with other widely used machine learning techniques for gesture recognition. The results indicates that approximate string matching is able to recognize gestures with highly acceptable accuracy and high performance. We have summarized the results in a radar chart (Fig. 10) to visually evaluate and compare the different techniques, when 12 different gestures are concurrently recognized. This kind of chart is useful to display observations simultaneously with an arbitrary number of variables. Based on the WSM previously explained in Section 4.2, we can clearly observe that String-6 outperforms the rest of the techniques by considering the evaluation time; however, the accuracy obtained by this technique was lower than the other techniques, specially when the number of gestures increases (Fig. 10). Therefore, taking into account a balance between these metrics (A1, with $\alpha_1 = 0.75$ and $\beta_1 = 0.25$) ASM-3 becomes the best option. Furthermore, if we also take into account the evaluation memory (A2, with $\alpha_2 = 0.6$, $\beta_2 = 0.2$, $\gamma_2 = 0.1$, $\delta_2 = 0.05$, $\omega_2 = 0.05$) ASM-3 continues still being the best option. Although consumed memory and spent time during the training phase are not critical for all applications, some applications might require to be trained in real time. In those cases, we could consider the A4 indicator (with $\alpha_4 = 0.6$, $\beta_4 = 0.2$ and $\gamma_4 = 0.2$), which clearly shows that ASM-3 is again the best option, particularly when considering a high number of gestures. Another example involves taking into account a balance among all the indicators (A3, with $\alpha_3 = 0.6$, $\beta_3 = 0.2$ and $\gamma_3 = 0.2$), thus presenting the following rankings where ASM-3 is once more the best option: for 4 gestures (ASM-3 (0.97), String-6 (0.95), Markov-3 (0.95), Procrustes (0.92), DTW (0.6)), for 8 gestures (ASM-3 (0.94), Markov-3 (0.85), String-6 (0.83), Procrustes (0.77), DTW (0.59)), and finally for 12 gestures (ASM-3 (0.95), String-6 (0.80), Markov-3 (0.78), Procrustes (0.72), and DTW (0.59)).

To sum up, the results obtained by our approach are promising. However, in order to generalize the results, we need to consider a number of threats to validity. Regarding accuracy, we need to consider the correctness of the gestures used for training and recognizing because we have split each gesture instance and discarded those gestures performed incorrectly. Thus, the technique did not confuse gestures and better recognition rates were

obtained. Despite this issue, we have included in the experiments gestures performed by people with different skills and different body builds. Another point to be taken into account is the joints selected in each case to train the techniques; we have selected the joints that we considered joints of interest and allowed the techniques to improve their accuracy. The importance of this issue stems from the fact that selecting different joints might produce different results. Concerning the performance, we should consider the hardware and the concurrency used in the experiments. We have run the experiments in a computer with specific characteristics, as detailed in Section 4.4, using a single thread in order to accurately measure the consumed memory and the spent time. However, using different computers or mobile devices with multiple threads might obtain different performance results.

5. Conclusions

In this work, we have presented a lightweight approach for recognizing gestures with Kinect. The main contribution of the proposed approach is the utilization of string matching for gesture recognition; in this combination, the gestures are encoded as sequences of characters and the recognition is performed by applying string matching. Specially, we used the approximate string matching algorithm based on dynamic programming, which provides support to efficiently recognize gestures. To encode gestures as sequences of characters, our approach applies the *k*-means algorithm for discretizing each 3D point in single digits, each of which corresponds to one cluster. Particularly, we have applied *k*-means with three clusters, however, other different numbers of clusters could improve the recognition accuracy in other gesture datasets.

To assess the effectiveness of our approach, we have used he public MSRC-12 Kinect gesture dataset, and compared the approach with other state-of-the-art gesture recognition techniques, such as Dynamic Time Warping, Procrustes Analysis, Markov Chains, and Hidden Markov Models. The results have shown that our approach was able to obtain better balance between accuracy and performance, in terms of consumed memory and spent time, than the compared techniques. Particularly, ASM-3 reached 98% accuracy on average when recognizing 12 gestures concurrently, and reduced the recognition consumed memory by 99.5% and the recognition spent time by 15.68% regarding DTW, which was the most accurate technique (99% of accuracy). Moreover, we have created a variant of our approach, which uses exact matching instead of approximate string matching. This variant was able to obtain 92% of accuracy when recognizing 4 gestures concurrently, outperforming the remaining techniques in term of consumed memory and spent time.

Nonetheless, our approach still needs some improvements, which we expect to address in future works. Firstly, we will work on providing an automatic selection of joints of interest, since the user must select the best body joints for recognizing an intended gesture. Secondly, we will focus on including alternative encoding algorithms and other clustering algorithms such as Partitioning Around Medoids (PAM) [46,47], in order to find a solution to the mislabeling of gestures that are considerably similar, thus improving the recognition accuracy. Thirdly, we propose to deal with the *k*-means initialization problem by using the MinMax *k*-means algorithm, which assigns weights to clusters relative to their variance and optimizes a weighted version of the *k*-means objective. Experiments have assessed the effectiveness of the approach and its robustness over wrong and weakly supported initializations; for this reason, we consider this approach viable and suitable to be applied in an enriched version of our approach [42].

Overall, we believe that our approach will enable efficient body gesture recognition not only in classical natural user interfaces, but also in new mobile devices that include a depth sensor, wherein cpu, memory, and energy consumption are critical factors in such limited resource devices.

Acknowledgment

We acknowledge the financial support provided by ANPCyT through grant PICT 2011-0080.

References

[1] J. Boehm, Natural user interface sensors for human body measurement, in: ISPRS – International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences XXXIX-B3, 2012, pp. 531–536.
[2] E. Suma, B. Lange, A. Rizzo, D. Krum, M. Bolas, Faast: The flexible action and articulated skeleton toolkit, in: 2011 IEEE Virtual Reality Conference (VR), 2011, pp. 247–248.
[3] F. Kistler, B. Endrass, I. Damian, C. Dang, E. André, Natural interaction with culturally adaptive virtual characters, J. Multimodal User Interfaces 6 (1–2) (2012) 39–47.
[4] V. Thiruvarudchelvan, T. Bossomaier, Towards realtime stance classification by spiking neural network, in: The 2012 International Joint Conference on Neural Networks (IJCNN), 2012, pp. 1–8.
[5] A. Bleiweiss, D. Eshar, G. Kutliroff, A. Lerner, Y. Oshrat, Y. Yanai, Enhanced interactive gaming by blending full-body tracking and gesture animation, in: ACM SIGGRAPH ASIA 2010 Sketches, ACM, New York, 2010, p. 34.
[6] X. Yang, Y. Tian, Eigenjoints-based action recognition using naïve-Bayes-nearest-neighbor, in: CVPR Workshops, IEEE, Providence, USA, 2012, pp. 14–19.
[7] S. Bhattacharya, B. Czejdo, N. Perez, Gesture classification with machine learning using Kinect sensor data, in: 2012 Third International Conference on Emerging Applications of Information Technology (EAIT), 2012, pp. 348–351.
[8] C. Waithayanon, C. Aporntewan, A motion classifier for Microsoft Kinect, in: 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), 2011, pp. 727–731.
[9] M. Parizeau, N. Ghazzali, J.-F. Hébert, Optimizing the cost matrix for approximate string matching using genetic algorithms, Pattern Recognit. 31 (4) (1998) 431–440.
[10] S. Salvador, P. Chan, Toward accurate dynamic time warping in linear time and space, Intell. Data Anal. 11 (5) (2007) 561–580.
[11] F. Petitjean, A. Ketterlin, P. Gançarski, A global averaging method for dynamic time warping, with applications to clustering, Pattern Recognit. 44 (3) (2011) 678–693.
[12] A. Ross, Procrustes Analysis, Course Report, Department of Computer Science and Engineering, University of South Carolina, 2004.
[13] K. Lange, Finite-state Markov chains, in: Numerical Analysis for Statisticians, Statistics and Computing, Springer, New York, 2010, pp. 503–526.
[14] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[15] P.R. Cavalin, R. Sabourin, C.Y. Suen, A.S.B., Jr., Evaluation of incremental learning algorithms for {HMM} in the recognition of alphanumeric characters, Pattern Recognit. 42 (12) (2009) 3241–3253 (New Frontiers in Handwriting Recognition).
[16] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, New York, 1997.
[17] S. Fothergill, H. Mentis, P. Kohli, S. Nowozin, Instructing people for training gestural interactive systems, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12), ACM, New York, NY, USA, 2012, pp. 1737–1746.
[18] M. Ziaeefard, R. Bergevin, Semantic human activity recognition: a literature review, Pattern Recognit. 48 (8) (2015) 2329–2345.
[19] D. Weinland, R. Ronfard, E. Boyer, A survey of vision-based methods for action representation, segmentation and recognition, Comput. Vis. Image Underst. 115 (2) (2011) 224–241.
[20] P. Turaga, R. Chellappa, V.S. Subrahmanian, O. Udrea, Machine recognition of human activities: a survey, IEEE Trans. Circuits Syst. Video Technol. 18 (11) (2008) 1473–1488.
[21] C.H. Lim, E. Vats, C.S. Chan, Fuzzy human motion analysis: a review, Pattern Recognit. 48 (5) (2015) 1773–1796.
[22] M. Barnachon, S. Bouakaz, B. Boufama, E. Guillou, Ongoing human action recognition with motion capture, Pattern Recognit. 47 (1) (2014) 238–247.
[23] S. Althloothi, M.H. Mahoor, X. Zhang, R.M. Voyles, Human activity recognition using multi-features and multiple kernel learning, Pattern Recognit. 47 (5) (2014) 1800–1812.
[24] J. Wang, Z. Liu, Y. Wu, J. Yuan, Learning actionlet ensemble for 3d human action recognition, IEEE Trans. Pattern Anal. Mach. Intell. 36 (5) (2014) 914–927.
[25] Z. Zhang, Microsoft Kinect sensor and its effect, IEEE Multimed. 19 (2) (2012) 4–10.

[26] J. Han, L. Shao, D. Xu, J. Shotton, Enhanced computer vision with Microsoft Kinect sensor: a review, IEEE Trans. Cybern. 43 (5) (2013) 1318–1334.
[27] J. Suarez, R.R. Murphy, Hand gesture recognition with depth images: a review, in: 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, IEEE, Paris, 2012, pp. 411–417.
[28] M. Tang, Recognizing Hand Gestures with Microsoft Kinect, Department of Electrical Engineering of Stanford University, Palo Alto, 2011.
[29] Z. Ren, J. Meng, J. Yuan, Z. Zhang, Robust hand gesture recognition with Kinect sensor, in: Proceedings of the 19th ACM International Conference on Multimedia (MM'11), ACM, New York, NY, USA, 2011, pp. 759–760. http://dx.doi.org/10.1145/2072298.2072443.
[30] K.K. Biswas, S.K. Basu, Gesture recognition using Microsoft Kinect ®, in: 2011 5th International Conference on Automation, Robotics and Applications (ICARA), IEEE, Wellington, 2011, pp. 100–103.
[31] K. Ponto, J. Kohlmann, R. Tredinnick, Dscvr: designing a commodity hybrid virtual reality system, Virtual Real. 19 (1) (2015) 57–70.
[32] Y. Gu, H. Do, Y. Ou, W. Sheng, Human gesture recognition through a Kinect sensor, in: 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), IEEE, Guangzhou (China), 2012, pp. 1379–1384.
[33] S. Celebi, A.S. Aydin, T.T. Temiz, T. Arici, Gesture recognition using skeleton data with weighted dynamic time warping, in: VISAPP (1), 2013, pp. 620–625.
[34] F. Jiang, S. Zhang, S. Wu, Y. Gao, D. Zhao, Multi-layered gesture recognition with Kinect, J. Mach. Learn. Res. 16 (1) (2015) 227–254.
[35] T. Hachaj, M.R. Ogiela, Human actions recognition on multimedia hardware using angle-based and coordinate-based features and multivariate continuous hidden Markov model classifier, Multimed. Tools Appl. (2015) 1–21.
[36] R. Ibañez, A. Soria, A. Teyseyre, M. Campo, Easy gesture recognition for Kinect, Adv. Eng. Softw. 76 (0) (2014) 171–180.
[37] R. Slama, H. Wannous, M. Daoudi, A. Srivastava, Accurate 3d action recognition using learning on the Grassmann manifold, Pattern Recognit. 48 (2) (2015) 556–567.
[38] P. Hong, M. Turk, T.S. Huang, Constructing finite state machines for fast gesture recognition, in: Proceedings of the 15th ICPR, 2000, pp. 691–694.
[39] T. Stiefmeier, D. Roggen, G. Tröster, Gestures are strings: efficient online gesture spotting and classification using string matching, in: Proceedings of the ICST 2nd International Conference on Body Area Networks (BodyNets'07), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Belgium, 2007, pp. 16:1–16:8.
[40] J. Wang, Z. Liu, Y. Wu, J. Yuan, Learning actionlet ensemble for 3d human action recognition, IEEE Trans. Pattern Anal. Mach. Intell. 36 (5) (2014) 914–927.
[41] G. Navarro, M. Raffinot, Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences, Cambridge University Press, New York, NY, USA, 2002.
[42] G. Tzortzis, A. Likas, The minmax *k*-means clustering algorithm, Pattern Recognit. 47 (7) (2014) 2505–2516.
[43] P.A.V. Hall, G.R. Dowling, Approximate string matching, ACM Comput. Surv. 12 (4) (1980) 381–402.
[44] D. Michie, D.J. Spiegelhalter, C.C. Taylor, J. Campbell (Eds.), Machine Learning, Neural and Statistical Classification, Ellis Horwood, Upper Saddle River, NJ, USA, 1994.
[45] E. Triantaphyllou, K. Baig, The impact of aggregating benefit and cost criteria in four MCDA methods, IEEE Trans. Eng. Manag. 52 (2) (2005) 213–226.
[46] S. Theodoridis, K. Koutroumbas, Pattern Recognition, 3rd ed., Academic Press, Inc., Orlando, FL, USA, 2006.
[47] J.-P. Mei, L. Chen, Fuzzy clustering with weighted medoids for relational data, Pattern Recognit. 43 (5) (2010) 1964–1974.

Rodrigo Ibañez received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2012, and is currently pursuing the Ph.D. degree in Computer Science at the same University. Since 2013, he has been part of ISISTAN Research Institute, UNICEN. His research interests include machine learning algorithms, human activity recognition, and automated web services composition. Mr. Ibañez has obtained a scholarship from FONCyT to complete his doctoral studies.

Álvaro Soria received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2001, and the Ph.D degree in Computer Science at the same university in 2009. Since 2001, he has been part of ISISTAN Research Institute (CONICET – UNICEN). His research interests include software architectures, quality-driven design, object-oriented frameworks and fault localization.

Alfredo Teyseyre received a Ph.D. degree in Computer Science, a Master degree in Systems Engineering, and a Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2010, 2001, and 1997, respectively. Since 1997, he has been part of ISISTAN Research Institute, UNICEN. Currently he is an Adjunct Professor at Computer Science Department of the UNICEN University at Tandil, Argentina. His research interests include software visualization, information visualization, software architecture and frameworks, and natural user interfaces.

Guillermo Rodriguez received the Ph.D degree in Computer Science at the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2014. Since 2008, he has been part of ISISTAN Research Institute (CONICET – UNICEN). Currently he is a Teaching Assistant at Computer Science Department of the UNICEN University at Tandil, Argentina. His research interests include software engineering, virtual reality and games for education.

Marcelo Campo received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 1988 and the Ph.D degree in Computer Science from Instituto de Informática de la Universidad Federal de Rio Grande do Sul (UFRGS), Brazil, in 1997. He is currently an Associate Professor at Computer Science Department and Director of the ISISTAN Research Institute (CONICET – UNICEN). His research interests include intelligent aided software engineering, software architecture and frameworks, agent technology and software visualization.