Advances in Engineering Software 76 (2014) 171–180

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Easy gesture recognition for Kinect

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

ISISTAN Research Institute (CONICET–UNICEN), Campus Universitario, Paraje Arroyo Seco, Tandil, Buenos Aires, Argentina

ARTICLE INFO

Article history:
Received 11 December 2013
Received in revised form 26 June 2014
Accepted 5 July 2014

Keywords:
Natural user interfaces
Gesture recognition
Machine learning
Kinect
Human-computer interaction
Gesture-recognition framework

ABSTRACT

Recent progress in entertainment and gaming systems has brought more natural and intuitive human–computer interfaces to our lives. Innovative technologies, such as Xbox Kinect, enable the recognition of body gestures, which are a direct and expressive way of human communication. Although current development toolkits provide support to identify the position of several joints of the human body and to process the movements of the body parts, they actually lack a flexible and robust mechanism to perform high-level gesture recognition. In consequence, developers are still left with the time-consuming and tedious task of recognizing gestures by explicitly defining a set of conditions on the joint positions and movements of the body parts. This paper presents EasyGR (Easy Gesture Recognition), a tool based on machine learning algorithms that help to reduce the effort involved in gesture recognition. We evaluated EasyGR in the development of 7 gestures, involving 10 developers. We compared time consumed, code size, and the achieved quality of the developed gesture recognizers, with and without the support of EasyGR. The results have shown that our approach is practical and reduces the effort involved in implementing gesture recognizers with Kinect.

1. Introduction

Over the last years, controlling systems by gestures through Natural User Interfaces (NUI) has become a common practice in our daily lives [1,2]. In 2006, WiiMote changed the concept of remote control by allowing players to control games through hand movements in the 3D space [3]. Then in 2010, Microsoft Kinect for Xbox became the most popular device not only for its low cost but also for its simplicity. Kinect allows players to control games through full-body movement without using a remote control [4]. Indeed, it has promoted the development of new natural interaction applications.

Although Kinect is able to recognize the position of users' joints, developers are still left with the time-consuming and tedious task of recognizing gestures. More precisely, early Software Development Kits (SDKs) only provided interfaces and code samples that enabled developers to access sensor data in real time, such as RGB-D camera, microphones, and the 3D position of 20 body joints. Later, some tools proposed to augment the interfaces with a rule-based approach that relied on a set of parameters and thresholds on joint location to track movements [5,6]. This approach supports the creation of gestures by allowing developers to adjust gesture sensibility by means of threshold values, and then to link these gestures to specific actions. However, this has become an error-prone process that requires domain knowledge, experience, and effort to ad hoc define a set of rules or heuristics so as to recognize human body gestures, in particular when defining complex gestures. Therefore, techniques that easily adapt to these complex needs would be necessary.

In order to provide a more flexible and robust approach, we can see gesture recognition as a classification problem [7]. In this context, a classification problem consists in assigning one label or class to a gesture in such a way that it is consistent with the available data about the problem. For dealing with a classification problem, machine learning techniques can be applied. These techniques use a gesture training set, in which each gesture is labeled to generate a classifier. In turn, this classifier evaluates the similarity between a new gesture and each of the trained gestures, resulting in the label of the most similar gesture. Although these methods offer high correct-classification rates [8–10], developers still have to implement complex algorithms for including gesture recognition in their own applications.

In this context, we propose EasyGR (Easy Gesture Recognition),[1] a gesture-recognition tool that allows developers to define and recognize gestures without demanding from them specific knowledge

* 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), marcelo.campo@isistan.unicen.edu.ar (M. Campo).

[1] EasyGR Homepage http://www.isistan.unicen.edu.ar/?page_id=501.

http://dx.doi.org/10.1016/j.advengsoft.2014.07.005
0965-9978/

of machine learning algorithms. The user interface of EasyGR allows non-specialist users to record, edit, and store gestures, enabling them to easily create a new training set. Then, by using this training set, these users may train different machine learning techniques for gesture recognition. Particularly, EasyGR supports two techniques: Dynamic Time Warping (DTW) [11] and Hidden Markov Models (HMM) [12]. Therefore, the approach was validated by analyzing two factors: the techniques' **accuracy** and the **efforts** required to develop gesture-controlled applications using and not using EasyGR. We tested the techniques' accuracy, by using 7 different gestures with 80 samples for each gesture. The results showed correct-recognition rates of over 99%. Concerning the amount of effort involved, we asked 10 developers to implement a gesture controlled application using a rule-based approach, and then to re-implement the application using EasyGR. The comparison between the solutions of each approach showed that EasyGR can certainly reduce the efforts to develop gesture recognizers.

The remainder of this article is organized into four sections. Section 2 covers related works. Section 3 describes the kind of assistance provided by EasyGR through a motivating example. Section 4 discusses the experiments and results, along with the benefits of using EasyGR. Finally, Section 5 presents the conclusions of this work.

2. Related work

There is a vast amount of literature on gesture recognition from human body movements captured by video cameras. For a review of the state of the art in human movement recognition in general, see [13–15,7,16]; for hand gestures see [17,2]; for facial expressions see [1]; for a review of recent Kinect-based computer-vision algorithms and applications see [18].

However, the development of natural interaction interfaces based on video cameras was found to be significantly difficult and therefore used mainly in certain specialized applications [4]. Recently, this situation has changed as a result of the broad availability of new 3D depth cameras, such as Microsoft Kinect, which promote the development of natural interaction applications in many domains among much larger audiences [18]. This device, composed of a Red–Green–Blue camera coupled with a Depth Sensor and a processing module, is able to estimate the movements of various body parts. Several SDKs, such as Microsoft Kinect SDK,[2] OpenNI[3] and OpenKinect,[4] provide Application Programming Interfaces (APIs) to enable developers to access sensor data in real time, such as RGB-D camera, microphones, and the 3D position of body joints. In spite of these APIs completeness, they still demand from developers remarkable efforts to recognize gestures.

Some works have proposed a rule-based approach that relies on a set of parameters and thresholds on joint location to track movements [5,6]. For example, FAAST [5] supported the creation of gestures by allowing developers to adjust gesture sensibility by means of threshold values (e.g., "RightHand.y > Head.y + 0.5") and then to map these gestures to key and mouse events to control arbitrary applications via full body natural interaction. Similarly, the FUBI framework [6] is able to describe a richer set of gestures by giving more complex configuration options in an XML-based definition language. Conversely, this kind of approach demands developers noticeable effort to ad hoc define and test a set of rules or heuristics in order to recognize human body gestures. Additionally, making rules to recognize complex gestures like a Smash in a tennis game becomes impractical.

Other attempts successfully adapted machine learning techniques -previously applied and extensively studied in the computer vision community- to gesture recognition using Kinect's skeletal data [8–10]. For example, Bhattacharya et al. applied Support Vector Machines (SVM) and Decision Trees (DT) to recognize aircraft gestures used in the military Air Force [8]. Another approach successfully applied an algorithm based on Dynamic Time Warping (DTW) [9]. Although these methods were successfully instantiated for certain applications with high correct-classification rates, developers had to implement complex algorithms and also perform the training process in an ad hoc way. In particular, Fothergill et al. addressed the problem of collecting gesture datasets to improve the accuracy and performance of the gesture-recognition system based on machine learning algorithms [10].

Wölfel [19] developed Kinectic Space, a tool similar to EasyGR, which makes it possible to record and automatically recognize customized gestures using DTW. However, some substantial differences exist between Kinetic Space and EasyGR. First, EasyGR provides not only the recording and recognition of complex gestures but also an object-oriented framework that helps developers to increase the quality and productivity of implementing gesture-controlled applications. In particular, our framework provides a better solution in terms of flexibility and adaptability. For example, new recognition algorithms can be easily added and different toolkits can be supported. Unlike Kinectic Space, which uses one sample of the gesture as a training set, our training process is based on n-samples that enable EasyGR to recognize variants of the gestures.

In this context, our goal is to allow developers to define and recognize gestures without demanding from them specific knowledge of machine learning algorithms and to reduce development efforts. In short, the main difference between the studies described above and our approach lies in that we addressed the experimental evaluation of the approach not only in terms of the accuracy of the recognition techniques, but also in terms of the development effort in practice.

3. Easy gesture recognition

When a person faces Kinect, it detects his/her body contour and identifies the position of 20 body joints in a 3D space (x, y, z). As the person moves in front of Kinect, it keeps track of the positions of each joint. These positions are calculated 30 times per second and packaged in a skeleton model called 'stick model'. Each stick model represents the positions of the 20 joints at a given time, and subsequent stick models, which contain the movements of the joints during a period of time, represent a gesture. Therefore, by using a collection of stick models, third-party applications can process the human body movements and provide natural interaction.

However, while Kinect works well for simple human gestures, such as a hand swipe motion, it fails to recognize complex gestures, such as a Smash in a tennis game, from a collection of stick models. To augment Kinect's gesture-recognition capabilities, developers can follow one of two more complex and robust approaches. The first approach consists in creating rules that specify a set of conditions on the positions of some body joints that indicate an intended gesture. This rule-based solution is good at recognizing simple gestures, but more complex gestures require the specification of several rules for identifying each change in the position of the joints. The second approach is a more flexible alternative and consists in applying machine learning techniques to the collection of stick models so as to learn and consequently identify a gesture.

[2] http://www.microsoft.com/en-us/kinectforwindows/.
[3] http://www.openni.org/.
[4] http://openkinect.org/.

However, developers must learn how to implement and adapt these techniques for gesture recognition.

In this context, we present EasyGR, a tool which assists the developer in recognizing gestures to further facilitate the task of implementing NUI applications. The aim of EasyGR is to retrieve the skeleton joints from Kinect and infer meaning from the skeleton movements in order to identify a gesture. Fig. 1 shows a schematic view of the proposed tool.

To recognize gestures from the skeleton movements, EasyGR works in two application modes: the training mode and the recognition mode. The former consists in exercising the gesture-recognition techniques with a set of potential gestures of the same movement so that it can be recognized. The training workflow starts when the trainer stands in front of Kinect and performs a sample of the gesture. When he/she does so, EasyGR stores the movements in a buffer as a collection of stick models, which represents an example of a gesture to be recognized. Next, EasyGR moves each stick model in the collection to the center of the Kinect detection field and then normalizes all the body joints in the stick model by using the distance between the neck and the spine. Thus, EasyGR is able to manage gestures performed in different places inside the field and to reduce the impact that different body builds of users have on the recognition of gestures. Having centered and normalized all the stick models, EasyGR exercises one of the supported recognition techniques, thus generating a reference value that modifies the acceptance threshold for the gesture. This threshold is determined by considering the two training samples of the same gesture that differ the most, and it is calculated in two different ways depending on the technique used (DTW or HMM). This is explained in detail in Subsection 3.3 below. It is worth mentioning that all the gestures used for training have to be manually segmented and checked for correctness by using the edition capabilities of EasyGR (More details can be found in Subsection 3.2).

Once the recognition techniques have been trained, EasyGR is ready to work in recognition mode. This working mode involves recognizing the gestures performed by the user as one of the trained gestures. As in the training mode, when the user moves in front of Kinect, EasyGR temporarily stores the movements in the buffer. Here, the recognition mode differs from the training mode in that, rather than using a manually segmented gesture, EasyGR segments the buffer at runtime using as segmentation size the average duration of the gestures registered in the training samples. When a new stick model is added to the segmented buffer, as in training mode, EasyGR moves and normalizes the collection of stick models in the segment and executes the gesture-recognition technique to generate the corresponding reference value.

Here, instead of storing the reference value as a new example of a gesture, EasyGR compares the value with the stored acceptance threshold of each gesture. If the reference value falls inside the acceptance threshold, it is considered a match – i.e., the input gesture is recognized – and EasyGR notifies the application so that it can perform the corresponding action. This action was previously specified by the developer after training each gesture, i.e., the developer linked gestures with actions in the application to be executed upon the recognition of a gesture.

This way the developer does not need to know the details behind the techniques and can modify the gesture recognition by training EasyGR again for the new collection of joint movements. Therefore, the development time and complexity for this kind of application should be reduced.

3.1. Motivating example

In order to clarify how EasyGR works in practice, we now present the following example. It consists in developing a tennis game where the main goal is to recognize the Smash gesture performed with the right hand. A Smash is a powerful downward hit that sends the ball forcefully over the net. Fig. 2 shows six frames of a person performing the Smash gesture in front of Kinect. Each frame contains the stick model of the person which is represented by the 3D position (x, y, z) of the 20 body joints. The white joint indicates the position of the right hand $(x_{rhand}, y_{rhand}, z_{rhand})$ in each frame, and the red line indicates the progression of this position in the performance of the Smash. In the last frame, we can observe the whole

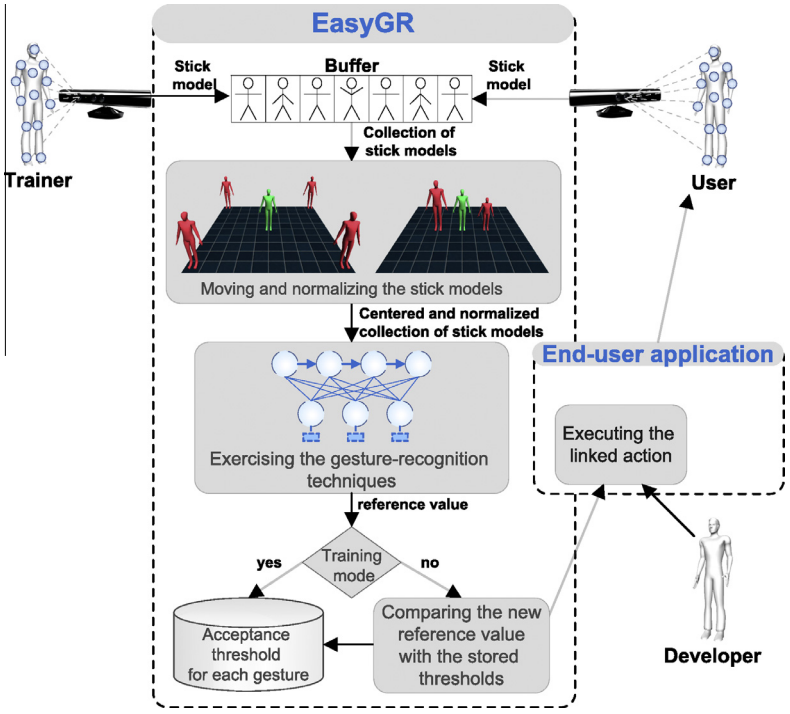


Fig. 1. Overview of EasyGR context.

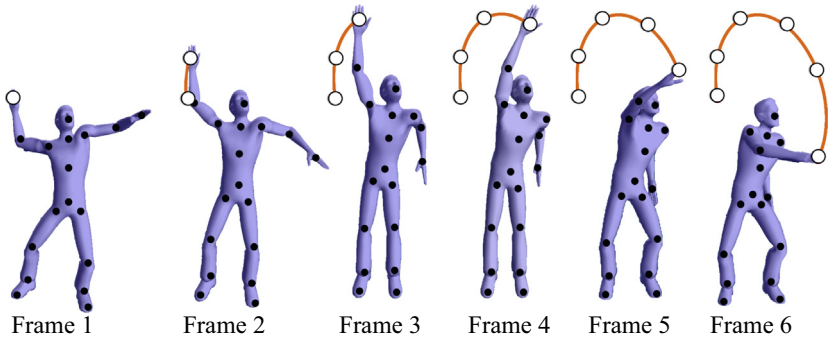


Fig. 2. Sequence of six stick models performing the Smash gesture.

progression of the right hand position called trajectory. We denote the trajectory of the right hand as $T_{rhand} = (x_1, y_1, z_1) \ldots (x_6, y_6, z_6)$. The trajectories of the body joints allow EasyGR to identify gestures.

For simplicity, let us assume that the trajectory corresponding to right hand is sufficient for EasyGR to identify the Smash gesture. To make gesture recognition possible, the trainer has to feed EasyGR with a set of samples of the Smash gesture. These samples should involve trajectories representing not only different ways of performing the same gesture but also different trainers with different body builds since the diversity of samples can improve the accuracy of the recognition techniques.

3.2. Moving and normalizing the stick models

Once diverse trajectories of the Smash have been performed, the next step consists in making the trajectories invariant to the trainer's position. As the trainer can be in different locations within the Kinect detection field, the 3D position of the Smash trajectories can drastically vary, making them unsuitable for comparison. To make the trajectories comparable, EasyGR translates the collections of stick models to the coordinates origin (0, 0, 0) as shown in Fig. 1.

This translation process involves the following steps. First, we calculate the *Centroid* of the spine trajectory using Eq. (1). 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. In this way, we obtain the distance and direction for translating the movement.

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

Next, we translate the collection of stick models so that the *Centroid* of the spine trajectory moves to the coordinates origin. We do this by subtracting the *Centroid* from each point of the collection of stick models (Eq. (2)). Thus, we obtain centered movements, which allow trajectories corresponding to the same gestures to be visually overlapped, i.e., the points of the trajectories are proximal.

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

The next step is to normalize the collection of stick models in order to reduce the impact of the different body builds on recognizing gestures. Thus, we normalize all the body joints in the stick model by using the distance between the neck and the spine as presented in [20]. Once we have centered and normalized all the collections of stick models, EasyGR is ready to be trained.

3.3. Exercising the gesture-recognition techniques

The next step in the training process is to exercise the recognition techniques with the centered and normalized trajectory set of the right hand representing the Smash gesture. For each trajectory, EasyGR generates a reference value which is used to calculate the acceptance threshold for the Smash. The threshold is determined by the two reference values that differ the most, i.e., the two trajectories that differ the most. When a new gesture is performed, EasyGR generates a new reference value for this gesture. If this value falls inside the acceptance threshold, then the gesture is recognized. To generate the acceptance threshold, EasyGR supports two techniques: Dynamic Time Warping (DTW) and Hidden Markov Models (HMM), each of which works differently.

3.3.1. Dynamic Time Warping

This algorithm finds the similarity between two time series by aligning them optimally. A time series is an ordered sequence of values measured at equally spaced time intervals. This sequence of values represents the trajectories of the right hand in our example. To align two trajectories, DTW iteratively stretches and shrinks the time axis so as to find the minimum distance between each pair of points in the trajectories. Applying DTW results in a distance value that measures the similarity between the trajectories.

After applying the algorithm between each pair of trajectories in the set, we obtain a model trajectory that represents the set and the acceptance threshold for the Smash. The model trajectory is the one that resembles the rest of the trajectories the most. We select it by adding all the distances from each trajectory to the rest and selecting the trajectory that gets the lowest distance. The upper limit of the acceptance threshold is given by the longest distance between the trajectories that differ the most. Therefore, we set the acceptance threshold between 0 and this value.

3.3.2. Hidden Markov Models

HMM is a way to represent stochastic processes, i.e., processes that model the aleatory behavior of one or more variables over time. In our example, the trajectories are modeled as a finite-state machine, where each state is the position of the right hand during the performance of the Smash. When a user performs a gesture, the main goal of HMM is to discover if the corresponding sequence of states is a possible state transition, i.e., is accepted by the model. As the right hand positions are continuous values, and therefore inappropriate for a finite-state representation, we need to apply a previous step that transforms all the points of the trajectories into a finite number of states.

This step consists in applying the k-means [21] algorithm to the training set of the Smash trajectories. The k-means algorithm groups the trajectory points in k numbered clusters, thus mapping each point with the number of the cluster and turning each

trajectory into a numeric sequence. To achieve this, first the algorithm randomly assigns all the points of the trajectories to one of the k clusters. Then, for each cluster, *k*-means calculates the centroid of all the points in the cluster by using Eq. (1).Then, *k*-means evaluates all the points and reassigns each point to the cluster whose centroid is nearest it, according to the proximity criteria (Eq. (3)). This criteria is the Euclidean distance between the point (P_i in the equation) and the cluster centroid (C_j in the equation). Having moved all the points to the corresponding cluster, the algorithm recalculates the centroid of each cluster. This process is repeated until no point movement is required, i.e., all the points are in the correct group.

$$D(P_i, C_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \quad (3)$$

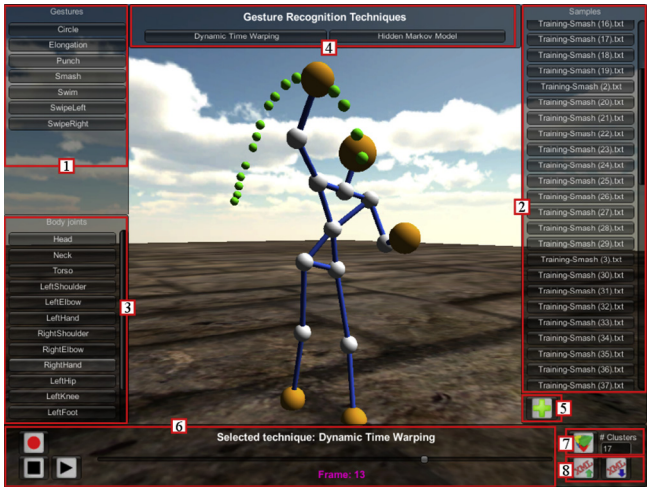
Eq. (3) calculates the Euclidean distance between the point P_i and the *Centroid* C_j.

Once we have applied *k*-means, we obtain a set of numbered clusters that represents the HMM states and all the trajectories in the training set represented as sequences of clusters. Now, we use all these sequences to train the HMM by applying the Baum–Welch algorithm [22]. This algorithm finds the most likely transition probabilities between states that best adapt the model to the training sequences.

Finally, we evaluate all the Smash sequences in the trained model to generate the acceptance threshold for the gesture. Evaluating a sequence means calculating the probability of this sequence belonging to the model. This is done by computing all the different paths of the model that can generate the sequence. After evaluating all the sequences, we select the minimum probability as the lower limit of the acceptance threshold. Therefore, we set the acceptance threshold between this value and 1.

3.4. EasyGR in action

To help trainers build the training set of the Smash and then train the techniques described above, we equipped EasyGR with a graphical interface as shown in Fig. 3. The interface allows trainers to define a gesture (1) and record a set of sample movements of this gesture (2). To train the techniques, the trainers can select the body joints (3) considered relevant in the recognition of the gesture (e.g., the right hand for recognition of the Smash). At this point, the trainer should analyze all the trained gestures that can be concurrently recognized and select the minimum number of joints to recognize the intended gesture. Having more than one joint to watch, EasyGR will be able to distinguish between two gestures that are too similar. Once the trainer has identified the required joints, she/he selects the recognition technique to be applied (4) and finally start the training by pressing the 'add training' button (5). In addition, the graphic interface allows trainers to visualize the stored gestures through the avatar in the central panel, control the playback (6), and apply *k*-means to the selected training set (7).



References:

1- List of different gestures	5- Start training
2- List of samples of the gesture	6- Gesture playback management
3- List of body joints	7- Execute k-means with # clusters
4- Available recognition techniques	8- Import and export the trained technique

Fig. 3. Graphic interface of EasyGR for training gesture-recognition techniques.

```
1  private Skeleton skeleton;
2  private int user = 1;
3  private GameObject avatar;
4
5  void Start() {
6    // User to be observed.
7    skeleton = EasyGR.SkeletonManager.getInstance()
         .getSkeleton(user);
8    // Initialization of the Smash.
9    Gestures Smash =
         Gestures.getGesture("Smash.xml");
10   Smash.RecognizedGesture += new
         Gestures.RecognizedGestureEventHandler(
         SmashRecognized);
11   Smash.startRecognizing();
12 }
13
14 void Update() {
15   // Update the avatar with the new positions
         taken from Kinect.
16   moveAvatar(skeleton, avatar);
17 }
18
19 public void SmashRecognized(Feedback feedback,
       Animation a) {
20   // Execute action for Smash.
21   executeSmash(avatar);
22 }
```

Listing 1. Recognizing the Smash by using EasyGR.

Once the selected technique has been trained, the last step consists in storing the result of the training to be used in the end-user application. To achieve this, the trainers press the export XML button (8) for EasyGR to create a configuration file that contains information such as the selected technique, the body joints used to determine the gesture, and the acceptance threshold values, among other data. It should be noted that developers do not have to train the techniques multiple times, i.e., every time they want a gesture to be recognized. EasyGR reads the values in the stored XML and uses them to determine if a new gesture is recognized.

In order to use the trained gesture in an end-user application, the developer needs to link the gesture recognition with an action inside the application. Let us assume the application that simulates the tennis game is being developed in C#, using Unity3D[5] as graphics engine. Listing 1 shows the code that a developer must include. This algorithm has the default C# script structure of Unity3D, which defines two states in the execution of the application: Start and Update methods. The Start method (line 5) is executed once and aims to define the variables that will be used over the execution of the application. This method involves three steps. The first step is to define the user that EasyGR will observe in order to recognize his/her gestures (line 7). With this instruction, the developer obtains

[5] http://unity3d.com/.

the reference to the user skeleton number 1. This skeleton contains the 3D positions of the body joints which are updated all the time by the Kinect SDK.

The second step involves loading the Smash gesture from the XML stored in the training phase (line 9). After that, the developer links the gesture recognition with an action, i.e., with the methods that EasyGR will execute when the Smash is recognized (line 10). Here, the link is made by means of the name of the method to be executed (SmashRecognized), which is defined by the developer. Note that EasyGR works asynchronously in relation to the application. When a gesture is recognized, EasyGR executes the linked method for the application to take over.

The third step of the Start method is to begin recognizing the gesture (line 11). From now on, as the player moves in front of Kinect, EasyGR adds each new stick model representing the player movement to a buffer. The buffer is a collection of stick models that implement a queue, in which the addition of a new stick model causes the removal of the old one from the buffer. We set the size of the buffer as the longest gesture used in the training step. Therefore, as the buffer is updated, EasyGR takes the stick models between 0 and the buffer size from the buffer and evaluates the recognition techniques with the models as input. If the value generate by the technique falls inside the acceptance threshold for the Smash gesture, EasyGR executes the linked method (line 19), and pauses the recognition process until all stick models that correspond to the recognized gesture are replaced in the buffer. The linked method should contain the actions that will be executed when the gesture is recognized, such as the avatar simulating the Smash and hitting the ball (line 21). Note that when EasyGR executes the linked method, it sends two parameters: Feedback and Animation. Feedback is an object that has information about the recognition and depends on the selected technique. If the technique is HMM, Feedback contains the probability of the recognition; if the technique is DTW, Feedback contains the distance applied to the trajectory to center it. On the other hand, Animation is the list of frames of the buffer used to evaluate the gesture-recognition technique.

Concurrently with the recognition of the Smash, the application displays its graphic environment. For this purpose, the developer writes the behavior of the displayed elements in the Update method (line 14). This method is executed periodically over the entire execution of the application. In our example, the developer wrote the visualization of the avatar movements by using the 3D positions of the body joints, which are updated by EasyGR.

When the application is running, the player moves in front of Kinect, thus updating the body joints and moving the avatar (line 16). Concurrently, EasyGR uses the body joints to evaluate the trained technique, trying to recognize the Smash. When this happens, EasyGR executes the linked method SmashRecognized, and the application executes the expected movements for the Smash.

Note that we created the example by using Unity3D engine; however, EasyGR can be used in any graphic engine that supports C# language.

4. Experimental results

This section describes the experiments to assess whether EasyGR helps developers to implement gesture-controlled applications. The objective of the first experiment was to assess the accuracy of the gesture-recognition techniques supported by EasyGR (Subsection 4.1). Therefore, we created a gesture training-set with seven gestures: Circle, Elongation, Swim, Smash, Punch, Swipe Right, and Swipe Left (Fig. 4). These gestures were selected because they involve movements of upper and lower body joints. Each gesture was performed 20 times by four people with different body builds and in different positions inside the Kinect detection field (a total of 560 samples). In addition, all samples were carefully checked for correctness. Thus, in order to estimate the accuracy of the different techniques, we applied the 10-fold cross-validation strategy [23]. The sample was randomly partitioned into 10 equal and

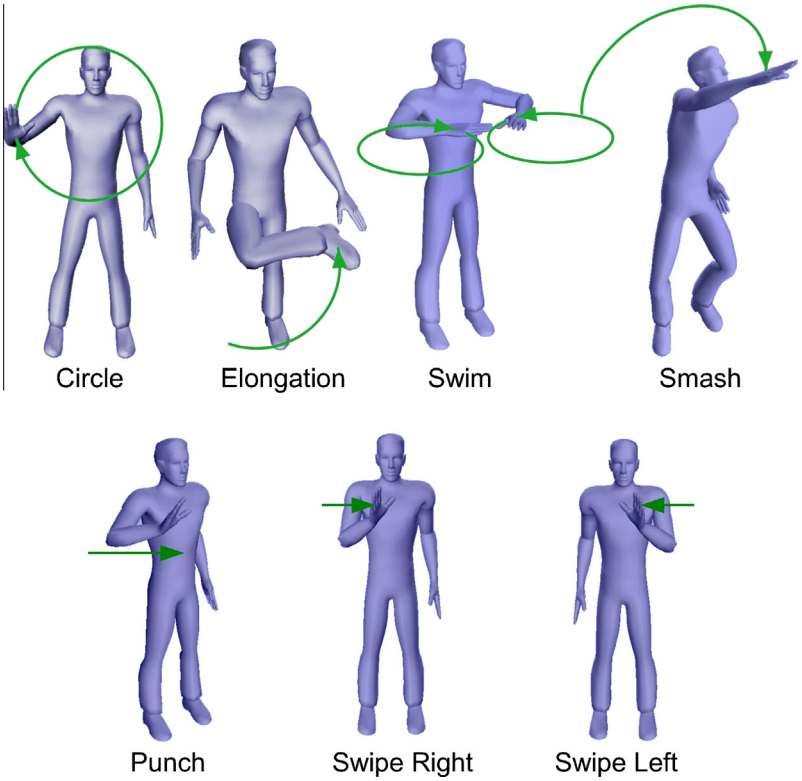


Fig. 4. Trajectory of the gestures used in the experiments.

balanced subsamples, nine of which were used for training and one for testing the model. This process was repeated 10 times with each of the 10 subsamples 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 mislabeling classification of gestures and perform a more detailed analysis than mere proportion of correct guesses. Finally, all results from the folds were averaged to produce a single estimation.

The second experiment involved 10 developers, who were given a two-phase assignment (Subsection 4.2). In the first phase, the developers designed an application controlled by the seven gestures described above by using the rule-based approach. In the second phase, they developed the same application by using EasyGR. To obtain an indicator of human effort, we computed and analyzed two metrics: non-blank, non-commented lines of code (NLoC) of methods implemented by a developer, and the time taken by a developer to implement a given gesture. We took these metrics on the resulting source codes in an attempt to assess the advantages of EasyGR over the rule-based approach in software maintenance.

Finally, we evaluated the quality of the resulting source-code when employing either the rule-based approach or EasyGR for recognizing gestures (Subsection 4.3). To compute the quality of each solution, we tested both solutions with a total of 210 gesture samples and compared their recognition rates.

4.1. Accuracy of the recognition techniques

To determine to what extent the number of samples used for training influenced the techniques' accuracy, we varied the number of samples among 20, 40, 60, and 80 samples for each gesture. Unlike DTW, HMM requires pre-defining the number of clusters to be used; therefore, to determine the influence of the number of clusters on the accuracy, we used 3 and 5 clusters. Fig. 5 illustrates the experimental results for DTW, HMM-3, and HMM-5. The number of samples is listed along the horizontal axis, while the technique's accuracy is listed along the vertical axis. From the bar chart, we can see that increasing the number of samples improves DTW's accuracy. For example, DTW's accuracy started at 0.95 when 20 samples were used and reached 0.991 when 80 samples were used. In addition, HMM's accuracy reached a peak of 0.99 for 3 and 5 clusters at 60 samples and then started to slightly decline to 0.97 in the case of 5 clusters. This decline can be explained by the fact that increasing the number of clusters improves the gesture fidelity, which makes the execution of the gesture more difficult to imitate. In other words, the imitation of the gesture must cross over each of the clusters to be recognized.

To visually evaluate the performance of both algorithms, we calculated a confusion matrix (Table 1) for DTW and HMM-3 using

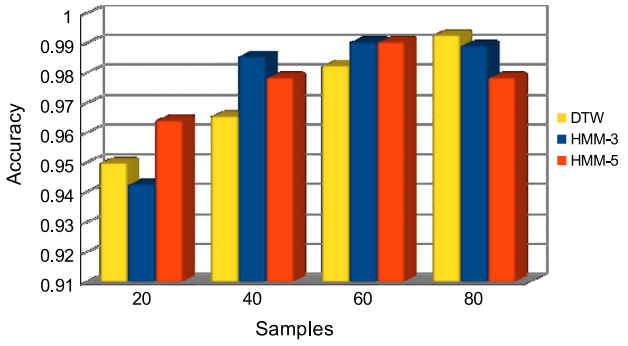


Fig. 5. Graph that compares the number of gestures correctly classified for each technique depending on the number of samples used for training.

Table 1
Confusion matrix of DTW and HMM-3 with 80 samples.

Known Class \ Predicted Class	Circle	Elongation	Swim	Smash	Punch	Swipe Right	Swipe Left
Circle	76	0	0	0	0	0	0
Elongation	0	80	0	0	0	0	0
Swim	0	0	80	0	0	0	0
Smash	0	0	0	80	0	0	0
Punch	4	0	0	0	80	0	1
SwipeRight	0	0	0	0	0	80	0
SwipeLeft	0	0	0	0	0	0	79

(a) DTW (80 samples / accuracy: 0.991)

Known Class \ Predicted Class	Circle	Elongation	Swim	Smash	Punch	Swipe Right	Swipe Left
Circle	80	0	0	0	0	0	0
Elongation	0	80	0	0	0	0	0
Swim	0	0	79	0	0	0	0
Smash	0	0	1	79	4	0	0
Punch	0	0	0	1	76	0	0
SwipeRight	0	0	0	0	0	80	0
SwipeLeft	0	0	0	0	0	0	80

(b) HMM-3 (80 samples / accuracy: 0.989)

80 samples. A confusion matrix shows how the predictions are made by the different techniques. In particular, the rows indicate the known class of the 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. All correct predictions are located along the diagonal of the Table, and the off-diagonal elements show the errors made. From this confusion matrix, we calculated three measures (accuracy, precision, and recall) in order to evaluate the performance of both techniques.

Accuracy is the overall correctness of the model and is calculated as the sum of correct classifications divided by the total number N of classifications, where the terms n_{ij} $(1 \leqslant i,j \leqslant k)$ correspond to the number of instances classified as class number i, when they actually belong to class number j, and k is the number of classes:

$$Accuracy = \frac{\sum_{i=1}^{k} n_{ii}}{N} \tag{4}$$

In particular, the accuracy of DTW and HMM-3 revealed a high correct-recognition rate of 0.991 and 0.989, respectively. In addition, accuracy should also be validated with precision and recall to perform a more detailed and rigorous evaluation of the performance of the classifiers.

Precision is a measure of the accuracy provided that a specific class has been predicted, that is, the correct classifications penalized by the number of incorrect classifications. It is defined by:

$$Precision_i = \frac{tp_i}{tp_i + fp_i} \tag{5}$$

Table 2
Precision and recall.

Class	Circle	Elongation	Swim	Smash	Punch	Swipe Right	Swipe Left
Precision							
DTW	0.95	1	1	1	1	1	0.99
HMM-3	1	1	0.99	0.99	0.95	1	1
Recall							
DTW	1	1	1	1	0.94	1	1
HMM-3	1	1	1	0.94	0.99	1	1

where tp_i and fp_i are the numbers of true positive and false positive predictions for the considered class i. In detail, precision is calculated from the confusion matrix as:

$$Precision_i = \frac{tp_i}{tp_i + fp_i} = \frac{n_{ii}}{n_{ii} + \sum_{j \neq i} n_{ji}} \quad (6)$$

Recall is a measure of the ability of a prediction model to select instances of a certain class from a data set, that is, the number of correct classifications penalized by the number of missed items. It is defined by the formula:

$$Recall_i = \frac{tp_i}{tp_i + fn_i} \quad (7)$$

where tp_i and fn_i are the numbers of true positive and false negative predictions for the considered class. In detail, recall is calculated from the confusion matrix as:

$$Recall_i = \frac{tp_i}{tp_i + fn_i} = \frac{n_{ii}}{n_{ii} + \sum_{j \neq i} n_{ij}} \quad (8)$$

From Table 2 we can see that both methods performed very well. The lowest precision was 0.95 for DTW in class Circle and 0.95 for HMM-3 in class Punch. The rest of the classes had a high precision rate. The lowest recall was 0.94 for DTW in class Punch and 0.94 for HMM-3 in class Smash, and the rest of the classes had a high recall rate.

4.2. Development efforts

For the second experiment, we selected 10 developers from the human–computer interaction course in the School of Exact Sciences at UNICEN, Argentina. This course was delivered to graduate and advanced students of Systems Engineering, who had programming experience. During the lectures, the developers were trained in machine learning for gesture recognition, rule-based approach, EasyGR, and application development by using Kinect SDK.

In order to start with the experiment session, we asked the developers to implement an application controlled by the seven gestures defined above, using a rule-based approach similar to [6]. Once the developers finished the assignment, we asked them to re-implement the gesture-recognition support by using EasyGR. In addition, we kept a record of the time consumed to implement each gesture recognizer. Note that, as the developers were familiar with the application domain and with both approaches prior to the two phases of the experiment, we considered that the order in which the two assignments were performed did not bias the experiment in favor of any approach.

Fig. 6 summarizes the results obtained by developing the gesture-recognition support for the seven gestures using the rule-based approach and EasyGR. Fig. 6a shows the NLoC needed for each approach. Note that we only considered the NLoC needed to implement the gesture-recognition support, leaving aside the line of code to implement the application behavior. Here, we can see a significant reduction in application size with EasyGR. Except for the NLoC of the Elongation gesture, the NLoC for recognizing only one gesture by using the rule-based approach is greater than the NLoC needed to implement all the gestures by using EasyGR. These improvements result from the gesture-recognition task being embedded in EasyGR, which requires the developer to implement only the code to be executed when the gesture is recognized.

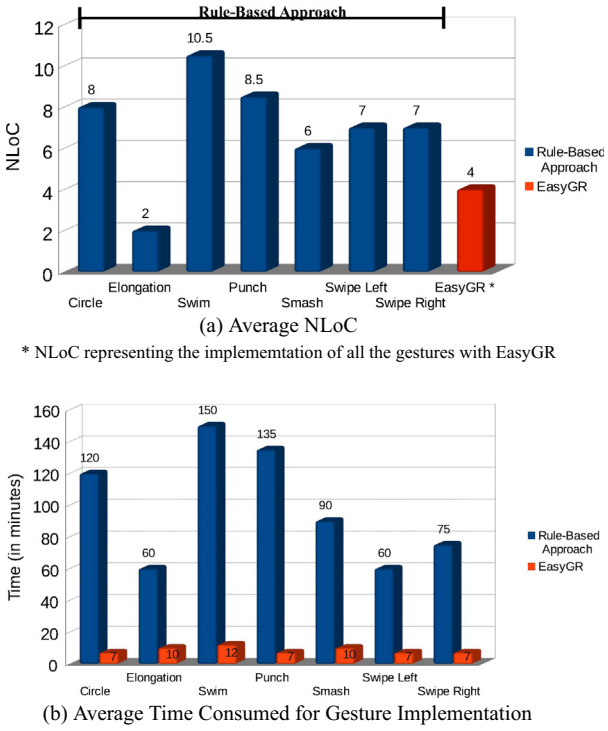


Fig. 6. Results comparing the Average NLoC (a) and the Average Time Consumed for Gesture Implementation (b) using and not using EasyGR.

With the rule-based approach, Swim, Circle, and Smash were the most complex gestures to be defined due to the changes in the direction of the movements. On the other hand, Punch, Swipe Left, and Swipe Right were the easiest gestures to be defined since each one involved movements of only one body joint in only one direction but the results show that these gestures required many rules. We attribute this to the fact that the rule-based approach requires a single line of code to define each gesture, but more lines to distinguish between gestures that are too similar and likely to be confused. For example, Swipe Left or Swipe Right and Swim are likely to be confused, since in both Swipe movements the rules are defined on the same body joints as in Swim. In addition, the fact that Swim involves movements that look similar to Swipe Left and Swipe Right together is a further source of confusion. Further, the two lines of code required by Elongation gesture may be explained by the fact that this gesture is the only one determined by the right-foot joint. To conclude, a developer wanting to include gesture recognition by using the rule-based approach must be concerned not only with the rules that define the gesture, but also with the rules that avoid confusing each gesture with the others being concurrently recognized. As a consequence, the greater the number of gestures that can be concurrently recognized, the bigger the effort required when using the rule-based approach.

Fig. 6b shows the time consumed per gesture implementation. The time consumed for EasyGR includes the time required to create

the training-set, link the application to EasyGR, and test the solution, while the time consumed for rule-based approaches includes the time required to develop and debug the rules and test the solution. Regarding the training time required by EasyGR, we took into account the time developers needed to create 25 samples for each gesture, because the developers realized that this number of samples was enough to achieve acceptable recognition rate. We can see from the chart that the participants using EasyGR required less time for developing a gesture than those not using EasyGR assistance. Moreover, the total time spent implementing the gesture-recognition support for the seven gestures by using EasyGR was shorter than the time spent implementing only one gesture by using the rule-based approach.

It is worth noting that the biggest effort when using EasyGR lies in creating the gesture training set. However, this effort – in terms of the time and the NLoC – is not as strenuous as that involved in defining the rules when using the rule-based approach.

4.3. Accuracy of the approaches in real-time gesture recognition

Finally, we compared the performance of EasyGR with that of the rule-based approach concerning accuracy in order to assess the code complexity vs. precision trade-off. Unlike Section 4.1, the purpose of the evaluation described in this section is not to assess the effectiveness of EasyGR when recognizing gestures, but to quantify the source code quality resulting from employing either rule-based or EasyGR for actually recognizing gestures. To achieve this, we compared both implementations by analyzing their precision in terms of correctly recognized real-time gestures.

For assessing the precision of the implementations, we randomly selected 30 samples of each gesture from the gesture set used in Section 4.1 above and then we reproduced each gesture in a real-time sequence so as to simulate a random Kinect input. To simulate a transition movement, we created and inserted a motionless gesture, which represents a user waiting to perform the next gesture, between gestures into the simulated sequence. Next, we used the sequence to exercise the implementations developed in both approaches. The number of gestures correctly recognized by each approach in real-time gesture recognition is summarized in Fig. 7.

The different gestures are listed along the horizontal axis, while the number of correctly recognized gestures is listed along the vertical axis. From the bar chart, we can see that except for the Punch gesture, in which the precision difference was marginal, using EasyGR achieved higher accuracy of the gesture recognition than the rule-based approach. In particular, EasyGR recognized 202 gestures from a total of 210 gestures, thus providing an accuracy of 0.96 in real-time gesture recognition, while the rule-based approaches recognized 177 from a total of 210 gestures reaching an accuracy of 0.84.

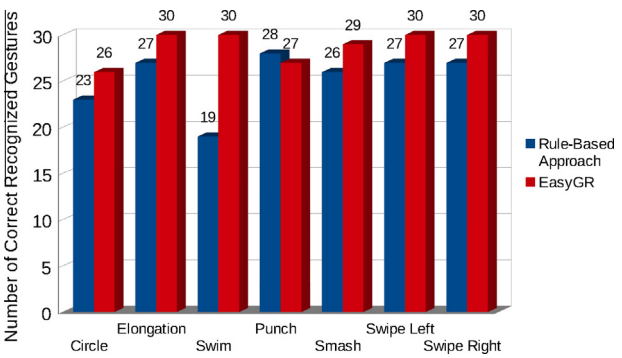


Fig. 7. Comparison between EasyGR and Rule-based approaches considering accuracy in real-time gesture recognition.

4.4. Lesson learned and threats to validity

In brief, EasyGR indeed facilitates the task of including gesture recognition in NUI applications, and the results of the evaluations of EasyGR are promising. During the experiments with different gestures and applications, EasyGR obtained correct-recognition rates of over 99% and reduced the development-time, size and complexity of implementing gesture-controlled applications. As we expected, the higher the number of samples used for training, the higher the accuracy of all the techniques.

Each technique has specific characteristics, advantages and disadvantages. The developer will choose the technique that best suits his/her needs. For example, DTW will be preferable when the priority of the developer is to know how the gesture is being recognized. In this way, the developer may obtain the distance from a new-user trajectory to the model trajectory and generate a feedback for the user, i.e., EasyGR informs the user how to improve the movement. This will be useful in applications like simulators. Instead, HMM will be the best choice if the application needs to vary the gesture fidelity. In this way the developer may be willing to reduce the gesture fidelity in order to obtain correct-recognition rates near 100% in applications that do not require a high gesture fidelity, like games. In such cases, the developer is responsible for finding a balance between gesture fidelity and the usability of the application.

Our study also highlights the importance of providing tools equipped with machine-learning techniques not only to ease the development of gesture recognizers but also to free the developer from the requirement of knowing the mathematical and logical details behind these techniques.

However, to generalize the results we need to consider a number of threats to validity. Regarding the technique's precision, we need to consider the body build of the person performing the gesture because we have not tested the tool with extreme features, e.g., very short or very tall people. In addition, we need to consider the correctness of each gesture because we performed the gestures in an ideal environment, i.e., we have not included the gestures performed in bad lighting, those performed away from the Kinect detection field, and those performed incorrectly. Concerning the development effort, we should bear in mind the developers' knowledge of gesture recognition and Kinect because we conducted the experiment with developers who were familiar with the projects and technologies used in the case studies. Anyway, this point deserves further studies since developers with different backgrounds, domain knowledge, or levels of expertise might achieve different solutions.

5. Conclusions

In this article we have presented a gesture-recognition tool to help developers include gesture recognition in NUI applications. In particular, EasyGR enables developers to create, train and recognize gestures. For the first two activities, EasyGR brings a graphic interface that allows non-specialist users to capture gestures and train the recognition techniques. EasyGR supports Dynamic Time Warping and Hidden Markov Models as recognition techniques. For the third, the developers have a template code for gesture recognition that they can include in their application, thus decreasing the development effort. Therefore, the main contribution of EasyGR is that it assists developers in the implementation of NUI applications. By encapsulating the gesture recognition, EasyGR reduces the complexity of managing the gesture data and the algorithms needed to build a NUI application.

The results of applying EasyGR in the case studies reported in this article are encouraging. We have evidence that EasyGR

contributes to reducing the code size and complexity of gesture controlled applications. Furthermore, we have obtained correct-recognition rates of over 99% with each techniques. In particular, we observed that increasing the number of clusters used in HMM improves the gesture fidelity but makes the gesture harder to imitate; thus the developer should find a balance between the gesture fidelity and the usability of the application.

Nonetheless, EasyGR still has some limitations and improvements that we expect to address in future versions of the tool. First, the gesture recognizer is unable to automatically adapt itself to the skills of the user. For instance, a developer creating a multilevel game would need a more restrictive recognizer every time the user completes each level. Although EasyGR does not have this feature, the developer can create different gestures to suit the user's skills. Second, we will focus on including new gesture-recognition techniques like Naïve-Bayes-Nearest-Neighbor [24] and String Matching [25].

Acknowledgements

We deeply thank the developers for their good predisposition towards the experiment, and the anonymous reviewers for their constructive comments about the article. We also acknowledge the financial support provided by ANPCyT through grant PICT-2011-0080.

References

[1] Mitra S, Acharya T. Gesture recognition: a survey. IEEE Trans Syst Man Cybernet Part C: Appl Rev 2007;37(3):311–24.
[2] Wachs JPJ, Kölsch M, Stern H, Edan Y. Vision-based hand-gesture applications. Commun ACM 2011;54(2):60–71.
[3] Lee J. Hacking the nintendo wii remote. IEEE Pervas Comput 2008;7(3):39–45.
[4] Zhang Z. Microsoft kinect sensor and its effect. IEEE Multimedia 2012:4–10.
[5] Suma E, Lange B, Rizzo A, Krum D, Bolas M. Faast: the flexible action and articulated skeleton toolkit. In: Virtual reality conference (VR), 2011. IEEE; 2011. p. 247–8.
[6] Kistler F, Endrass B, Damian I, Dang C, André E. Natural interaction with culturally adaptive virtual characters. J Multimodal User Interf 2012;6(1–2):39–47.
[7] Weinland D, Ronfard R, Boyer E. A survey of vision-based methods for action representation, segmentation and recognition. Comput Vis Image Underst 2011;115(2):224–41.
[8] Bhattacharya S, Czejdo B, Perez N. Gesture classification with machine learning using kinect sensor data. In: 2012 third international conference on emerging applications of information technology (EAIT); 2012. p. 348–51.
[9] Waithayanon C, Aporntewan C. A motion classifier for microsoft kinect. In: 2011 6th international conference on computer sciences and convergence information technology (ICCIT); 2011. p. 727–31.
[10] Fothergill S, Mentis H, Kohli P, Nowozin S. Instructing people for training gestural interactive systems. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI '12. New York (NY, USA): ACM; 2012. p. 1737–46.
[11] Salvador S, Chan P. Toward accurate dynamic time warping in linear time and space. Intell Data Anal 2007;11(5):561–80.
[12] Rabiner LR. Readings in speech recognition. San Francisco (CA, USA): Morgan Kaufmann Publishers Inc; 1990. Ch. A tutorial on hidden Markov models and selected applications in speech recognition. p. 267–96.
[13] Gavrila DM. The visual analysis of human movement: a survey. Comput Vis Image Underst 1999;73(1):82–98.
[14] Aggarwal J, Cai Q. Human motion analysis: a review. Comput Vis Image Underst 1999;73(3):428–40.
[15] Yu Q, Cheng HH, Cheng WW, Zhou X. Ch opencv for interactive open architecture computer vision. Adv Eng Software 2004;35(8–9):527–36.
[16] Turaga P, Chellappa R, Subrahmanian VS, Udrea O. Machine recognition of human activities: a survey. IEEE Trans Circ Syst Video Technol 2008;18(11):1473–88.
[17] Pavlovic V, Sharma R, Huang T. Visual interpretation of hand gestures for human–computer interaction: a review. IEEE Trans Pattern Anal Mach Intell 1997;19(7):677–95.
[18] Han J, Shao L, Xu D, Shotton J. Enhanced computer vision with microsoft kinect sensor: a review. IEEE Trans Cybernet 2013;43(5):1318–34.
[19] Wölfel M. Kineticspace – 3d – gestenerkennung für dich und mich. Konturen 2012;32:58–63.
[20] Lai K, Konrad J, Ishwar P. A gesture-driven computer interface using kinect. In: 2012 IEEE southwest symposium on image analysis and interpretation (SSIAI); 2012. p. 185–8.
[21] Jain AK. Data clustering: 50 years beyond *k*-means. Pattern Recogn Lett 2010;31(8):651–66.
[22] Ibe OC. 14 – hidden markov models. In: Ibe OC, editor. Markov processes for stochastic modeling. Oxford: Elsevier; 2013. p. 417–51.
[23] Michie D, Spiegelhalter DJ, Taylor CC, Campbell J, editors. Machine learning, neural and statistical classification. Upper Saddle River (NJ, USA): Ellis Horwood; 1994.
[24] Yang X, Tian Y. Eigenjoints-based action recognition using Naïve-Bayes-Nearest-Neighbor. In: CVPR workshops. IEEE; 2012. p. 14–9.
[25] Stiefmeier T, Roggen D, Tröster G. 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), ICST, Brussels, Belgium, Belgium; 2007. p. 16:1–:8.