A Markerless Gestural Interface for Avatar Control

April N. Suknot*[1] and Jessica K. Hodgins†[2]

[1]Department of Computer Science, University of New Mexico
[2]Department of Computer Science, Carnegie Mellon University

Abstract

This project serves as a gestural interface for searching a motion capture database. This system is intended to handle the gestures of a wide range of users employing algorithms that were trained on gestures identified in a user study. With this system, users can act out gestures in front of a natural interaction device, which will trigger different behaviors of an on-screen character. This inexpensive system captures movements using a Microsoft Kinect or similar sensor. A blob tracking method is used in conjunction with hidden Markov models (HMMs) to recognize the motions to be retrieved from a motion capture database and performed by an on-screen avatar.

1 Introduction

The system presented in this paper uses data from a Microsoft Kinect receiver to control an on-screen avatar based on recognized motions. When given a gesture that it recognizes, the system will search the Carnegie Mellon University motion capture database [1] to find a clip pertaining to that motion. The action will then be performed by a character rendered in the user interface.

1.1 Motivation

As video game consoles and devices created for these consoles continue to evolve, new ideas are conceived which allow users to interact with these systems in new ways. The Microsoft Kinect provides developers with access to visual information through an RGB camera and depth sensor. With this information, developers can create unique interfaces that allow people to make use of them to manipulate on-screen data in an inexpensive and easy to use setting.

1.2 Previous Work

There have been many projects in which puppeteering and motion capture databases have been used to facilitate on-screen animations. In [2], the authors created a system in which a user could specify a chain of motions in one of several different ways to control an on-screen avatar. They introduced a unique structure, a motion cluster, connecting similar clips from a motion capture database. Similar techniques are used with people acting out motions using an inexpensive marker-based motion capture system in [3] to create simulated motions. In [4], motions acted out with a doll are used to create on-screen animations, and in [5], motions performed by a puppet with sensors in key locations are used to retrieve motion capture data.

The Microsoft Kinect and similar devices have been used in research to create a variety of applications using puppeteering and gesture recognition. In [6], the Kinect is used to produce 3D models with hand-held puppets which are scanned by the Kinect and recorded in a database for subsequent recognition. These models can then be used to create stories or scenes acted out by a user with the puppets in front of a Kinect receiver. The authors of [7] also use a Kinect to track fingertips in the implementation of a virtual keyboard. They made use of blob tracking to monitor fingertip location and used a protocol to send fingertip information to various applications like a virtual keyboard and on-screen object manipulation. In [8], the Kinect was used to recognize various sign language gestures.

2 Methods

This system is built with three major components: hand tracking, gesture recognition, and playback of motion clips. These components will be discussed in the following subsections. In Section 2.1 we will first discuss the hardware setup of the system and how the design of the software was approached. Subsequently, in Section 2.2, approaches and methods used for hand tracking will be discussed. Methods used for gesture recognition and motion retrieval from the motion capture database will then be discussed in Section 2.3. Finally, the real-time rendering of the motion capture clips and the user interface will be discussed in Section 2.4.

2.1 Design and System Set Up

The hardware setup for this system is simply a single Microsoft Kinect or similar motion receiver whose data is accessed with OpenNI drivers [9]. The Kinect was the receiver used for testing the platform and has a resolution of 640x480 and a frame rate of 30hz [7]. The Kinect does have some distance constraints

*asuknot@cs.unm.edu
†jkh@cs.cmu.edu

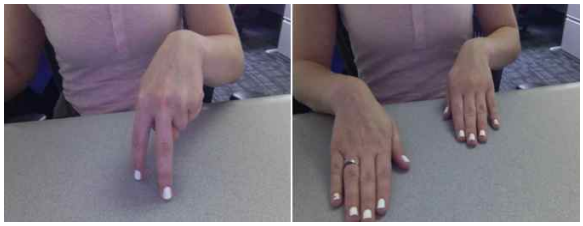

Figure 1: The image on the left shows the one-handed approach for acting out a marching gesture with their fingers, while the picture on the left shows a user performing the same gesture with the two-handed approach.

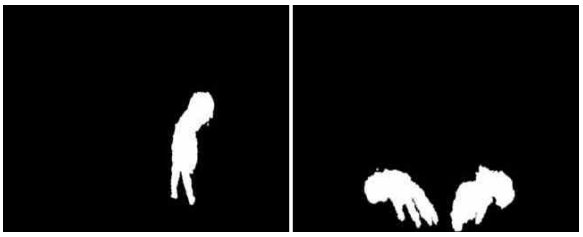

Figure 2: As in Figure 1, these images show the differences of the two considered approaches for gesture delivery, with one and two hands. However, this figure shows the two approaches as seen with depth segmentation.

for data reception as well. The user must maintain a distance of about three feet from the camera for best results. A user can have a distance of up to several meters from the receiver while still being tracked with reasonable accuracy. Moreover, our system requires that the user's hands be the closest objects to the camera for optimal performance. This constraint means that nothing can be in the Kinect's line of sight in front of the user's hands. Because our system recognizes gestures performed by the hands, it is best if the Kinect is on the same surface as that on which their hands are anchored when not performing a gesture.

To aid in the determination of which types of gesture performance users would be most comfortable with, a small user study of ten people was conducted to analyze preferences of people in different age groups. The youngest participant was seven years old and the oldest was sixty-three. Six of the participants were male and four were female. Each subject performed ten gestures corresponding to motions selected from the motion capture database in two different ways. The first method was a one-handed approach with fingers acting as the hands and legs. In the second approach, subjects used both hands in acting out motions with each hand representing a hand or a foot, depending on the motion. An example of this can be seen in Figure 1. There was an unanimous preference of gesturing with the two-handed approach over using fingers. However, there were some motions, such as the cartwheel, that were difficult to perform regardless of whether hands or fingers were used. We decided that the system should track two hands for motion retrieval. Five target gestures were chosen to be extracted from the database using hand motion: running, jumping, marching, dribbling a basketball and kicking.

2.2 Hand Tracking

The first step of hand tracking was to isolate the hands using depth segmentation. During the user study, some subjects were recorded with a Kinect to assess the quality of the depth segmentation and situations in which it can fail, as seen in Figure 2. The system proved to be rather successful with depth thresholding alone. The major cause of failures were when the subject's face would be captured by the camera because they leaned forward. Because of this skin color detection techniques like those described in [7] and [10] would have likely been ineffective for our purposes. As a result, the isolation portion of hand recognition was limited to depth segmentation. The responsibility of determining whether objects detected are the user's hands was then left to the blob detection process.

For blob detection and tracking, several approaches were evaluated and considered. Had we chosen to do the one-handed method in which the user gestures with his or her fingers instead of both hands, an approach like that used in [8] would have been employed. However, since we opted to go with the two-handed approach, a simpler blob detection algorithm was sufficient. In [7], a blob tracking library is used to track the trajectory of fingertips. Similarly, in [11], an external library is used for blob detection and is modified to track blobs based on their size. The approach in [11] was considered. However, for that implementation, blobs were captured with a webcam and maintained a relatively constant size and shape between frames. In our system, the depth data received through the Kinect can be noisy and it is common for blobs to change size and shape between frames. This also implies that the blobs could have drastically different locations between frames when calculated by an external source. In addition, we wanted to be able to track the blobs in three dimensions, which was not part of the functionality of some third-party libraries.

Furthermore, because our system is intended to be easily downloaded by users, dependence on external libraries was kept to a minimum. On a similar note, this system was created with the purpose of being portable on multiple operating systems, and some libraries have different dependencies for different systems and architectures. We observed that some libraries considered would not even function on certain architectures. With our system, the only dependencies are the OpenNI drivers [9] for the computer to be able to interface with an interaction device like the Kinect.

Because of these matters, along with the notion that we do not need to know the actual shape of the blob, an efficient and simplistic approach was taken for blob detection and tracking. To detect the blobs, a depth map is first created using depth segmentation based on a threshold that allowed objects the foreground to be the only objects captured. The closest object to the camera is found by identifying the minimum depth value that is recognized and an adjustable threshold is set from there to isolate the user's hands. This is the reason, as mentioned in

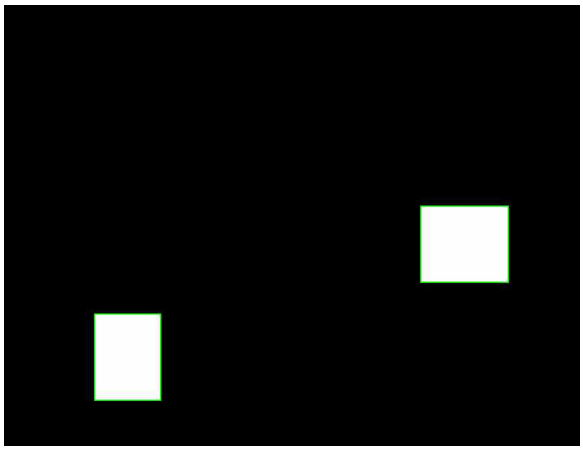

Figure 3: Subsequent to depth segmentation, bounding boxes are created around each blob to aid in keeping track of the position.

section 2.1, that the user must ensure that his or her hands are the closest objects to the receiver.

The depth map then contains the depth value of every pixel within the threshold and a zero otherwise. The depth map is then sent to the blob detection method which scans the frame for likely maximum and minimum edges of the blob for each frame. In each subsequent frame, the horizontal and vertical distance from the previous blobs are considered to decide if a pixel with a nonzero value in the depth map is within one of the blobs being tracked. This allows the algorithm to disregard noise or another object temporarily noticed by the sensor. The values in the x and y axis, along with the depth data reported by the Kinect are considered for each blob to create a bounding box around the blob as depicted in Figure 3. Through experimentation, it was found that the upper-left corner of the bounding box around the hand was typically the most consistent during gesturing. This is because the forearm of the user may appear at times beneath the hand during some gestures, ending up within the threshold. This causes the lower edge of the box to vary more. Because of this, the upper-left corner of the bounding box is used to keep track of the position of the blob in each frame.

As was noticed by the authors of [7], the frame rate of the Kinect coupled with the ability of a human to move their hands only a certain distance in between frame captures implies that there will be some overlap in blob positions over several frames. We take advantage of this to limit the amount of noise picked up by the receiver and to ensure that each location is accurate enough for gesture recognition. An average over several frames is taken to get the position of each blob in a short interval of time. Because the Kinect may pick up noise in one single frame which could make the blob much larger than in adjacent frames, if values are extreme compared to those in nearby frames, those values are discarded when calculating the average. Moreover, all frames in which only one blob, or no blobs are detected are ignored in the tracking. This heuristic allows the system to determine which direction each blob moved within each axis to be considered by the gesture detection algorithm.

2.3 Gesture Recognition and Motion Retrieval

The five target gestures that we chose have some similarities when performed with two hands. However, we found that the most noticeable differences are in the vertical component. The horizontal and depth components are used to determine the direction of movement, but the vertical component is used to resolve which gesture was performed by the user. To do so, this system uses a hidden Markov model (HMM) for each gesture to determine which motions to retrieve when a gesture is performed in front of the receiver. To implement the HMMs, we used the information and approach defined in [12] and described below.

The creation of the HMMs began with training each model on its respective gesture. This was done by first initializing the model, $\lambda = (A, B, \pi)$ with random values. In this model, A is an NxN matrix containing state transition probabilities, where N is the number of states in the model. B is an NxM observation probability matrix where M is the number of possible observations. Finally, π represents the N element vector which holds the initial state distribution.

With the model initialized, it was given a sequence of T observations, one for each of forty time steps. In our model, each observation represents the state of both hands when observed. Because we are considering the vertical position, each hand can be going up, going down, or staying still. The combinations of all positions for both hands yield $M = 9$ possible observations.

We then populated each model, $\lambda = (A, B, \pi)$ with values based on real data with an iterative approach known as the Baum-Welch algorithm. The approach begins with using the Forward-Backward, also known as the Viterbi algorithm to generate two matrices, α, β, and two values annotated by γ, one representing a gamma and the other a di-gamma. This method first employs a recurrence relation in the α, or forward pass.

$$\alpha_t(i) = [\sum_{j=0}^{N-1} \alpha_{t-1}(j)a_{ji}]b_i(O_t) \tag{1}$$

In equation 1, a_{ji} represents an element in matrix A at that index and $b_i(O_t)$ represents a value in matrix b in the row *i* and in the column representing observation *O* at time *t*. This recurrence relation begins with the base case, $\alpha_{\mathrm{o}}(i) = \pi_i b_i(O_{\mathrm{o}})$ for all *i* from *0* to *N-1*. This recurrence relation is computed for all *t*, from *1* to *T-1*.

Once the forward pass is complete, the backward, or β pass can be started. The β pass also uses a recurrence relation, which has the base case $\beta_{\mathrm{o}}(i) = 1$ for all *i* from *0* to *N-1* and is defined as

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \tag{2}$$

This recurrence relation is computed for all *t* from *T-2*, down to *0*. All elements in equation 2 are equivalent to those as described for equation 1. Finally, the gamma and di-gamma values can be computed. The gamma value is defined as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=0}^{N-1} \alpha_{T-1}(i)} \tag{3}$$

Finally, the di-gamma value is defined with

$$\gamma_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=0}^{N-1} \alpha_{T-1}(i)} \tag{4}$$

From here, the iterative method can update *A*, *B*, and π. To populate π, we compute the following for *i* from *0* to *N-1*.

$$\pi_i = \gamma_0(i) \tag{5}$$

To update *A*, we sum the following for *i* and *j* from *0* to *N-1*.

$$a_{ij} = \frac{\sum_{t=0}^{T-2} \gamma_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)} \tag{6}$$

Finally, to update the values in *B*, the following is computed for *i* from *0* to *N-1* and for *k* from *0* to *M-1*.

$$b_j(k) = \frac{\sum_{t=0}^{T-2} \gamma_t(j), where O_t = k}{\sum_{t=0}^{T-2} \gamma_t(j)} \tag{7}$$

The iterative algorithm returns to equation 1 if the probability of the observation performed in the training, given the model, $P(O|\lambda)$, does not increase by a set amount. The algorithm will also terminate if a maximum number of iterations have been performed. This probability is given by

$$P(O|\lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i). \tag{8}$$

For each of the five gestures, a model $\lambda = (A, B, \pi)$ was created and saved after training with many samples of given observations. Once these models are saved, the gesture recognition could be implemented. The system receives a sequence of observations, just as in the training. However, it now uses only the forward procedure, or α to calculate the matrix α, as in equation 1 for each model $\lambda = (A, B, \pi)$. After this step is complete, the system can compute the probability, $P(O|\lambda)$, as given in equation 8. This operations is computed for each model so that the model with the greatest probability can be recognized. The gesture associated with the model having the highest probability is then selected to be displayed to the user.

2.4 User Interface

When designing the user interface that would be employed by this system, the file type for the motion capture files was the first consideration. After reviewing several different file formats, the Biovision Hierarchy (BVH) file format was selected. This choice was made primarily because the file type itself contains the skeleton information for the character displayed on-screen as well as the orientation of each joint in each frame as described in [13]. This makes the file easy to read and understand and the full motion clip easy to keep track of when searching the database for a specific clip. The entire database was available in BVH format in easy to download bundles from [14].

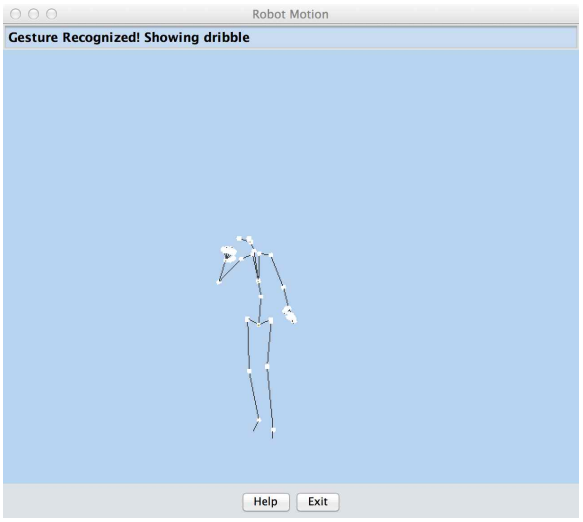

Figure 4: The user interface for communicating with users and displaying the gestures recognized that are performed.

For rendering the model, an open-source Java library, called JMocap [15], was modified and used as the focal point of our graphical user interface (GUI). Unlike other libraries considered for this system, the JMocap library was easily augmented into our system without requiring any additional dependencies or limiting the compatibility of this program with different operating systems. The original format of this motion capture viewer is in a GUI that has a large control panel which included options for the users to select videos. Since our interface is for retrieval of videos based on gestures recognized, much of this functionality was scaled down. In addition, some small changes were made to the functionality and color scheme to tailor the viewer more to our system.

In our interface, a user is prompted to perform a gesture. When a gesture is recognized, the system announces which gesture was recognized and a clip is pulled from the motion capture database and played on-screen with the motion capture viewer, as seen in Figure 4. The database is indexed with human-readable descriptions for each clip like ”forward dribble” and ”run left.” This index is used to search the database for keywords like the name of the gesture given. Direction measured by horizontal and depth positions of the blobs are used to filter the search for motions in that particular direction. The system finds and selects such a file by parsing the index, then uses the JMocap library to load the motion capture file and play the clip on-screen.

Gesture	Kicking	Running	Jumping	Dribbling	Marching
Kicking	54	14	8	22	0
Running	0	19	0	0	0
Jumping	1	2	24	4	0
Dribbling	3	2	20	23	0
Marching	42	63	48	51	100

Table 1: Confusion matrix totals for five twenty gesture tests using ten samples and four states. The gestures listed across the top represent the gesture given, the gestures along the left side of the table represent the gesture predicted by the models.

Gesture	Kicking	Running	Jumping	Dribbling	Marching
Kicking	57	14	5	20	0
Running	0	27	0	0	0
Jumping	1	2	45	3	0
Dribbling	4	2	13	29	0
Marching	38	55	37	48	100

Table 2: Confusion matrix totals for five twenty gesture tests using twenty samples and four states. The gestures listed across the top represent the gesture given, the gestures along the left side of the table represent the gesture predicted by the models.

3 Results

In implementing the HMM for gesture recognition, different variables were considered. Different numbers of samples used to train the model were tested to assess the quality of the results. Each gesture was trained on several separate sets, then training data for one set would be used as testing for the other sets. We tested five, ten, and twenty sample sets modeled with two through five states and nine states. Each was tested with an input of twenty gestures.

The marching gesture, was most predictable over all tests with the changing numbers of states, with an increase in performance with an increase in number of states. With two and three states marching had recognition rates between eighty and ninety percent over different tests. With four or more states, marching has a consistent hundred percent recognition rate. The kicking gesture also improves with state size from being near thirty percent with two states and above fifty percent with four states. When increased to nine states, kicking improved to eighty percent correct recognition. Some gestures, like running and jumping had decreased performance with increasing states. With three states, jumping was recognized correctly about ninety percent of the time and running about seventy percent of the time with two states. The performance was worse for both gestures when tested with more states. Due to this effect, a closer to median number of states seems to have the most positive results when considering all gestures.

Overall, tests using ten samples performed much better than tests using five samples for all states. For each gesture, there appeared to be a decrease in false positives, in which the incorrect gesture is recognized. There is also an apparent increase in successful recognition of each gesture. Some gestures, like kicking and dribbling did not improve as much as other gestures. There were some individual tests in which kicking or dribbling would have slightly worse performance with the sample size increase. It is presumed that this is a result of the similarity of these motions with marching could cause marching to be falsely identified more often in some cases. Overall however, there was a small improvement in the recognition performance with the sample size increase for the kicking and dribbling gestures. Running and jumping both had significant improvements when increasing the sample size. Increasing sample size did improve the recognition rate of marching for two and three states. For four or more states, marching stayed at a hundred percent recognition, regardless of sample size.

Marching was the gesture having the highest number of false positive results. The confusion matrices shown in Tables 1 and 2 provide an instance of the high rate of marching false identification depicting results from a series of tests using four states and sample sizes of ten in Table 1 and twenty in Table 2. Running is most commonly confused with kicking in addition to marching. However, both marching and kicking are seldom identified as running in lower numbers of states and were not identified as running with four or more states. Jumping and dribbling are sometimes mistaken for one another. With sample size increases, the false positives between dribbling and jumping have a tendency to decrease.

4 Conclusion and Future Work

This paper has introduced a system which allows a user to control an avatar on-screen by performing gestures in front of an inexpensive motion detection device like the Microsoft Kinect. The primary goal of this project was to find an inexpensive and user friendly way to allow people to use gestures to control a character. Many people are captivated by new and interesting ways to interact with the different types of computer systems that they use. For that reason, a system like this may have many applications in computer user interface design, animation, gaming, and other industries. There are aspects that should be modified in this system to make it even more accommodating and robust.

To make the system more fun and friendly for users, a motion graph similar to the ones described in [2] and [16] can be employed. Adding this functionality to the system can be used to keep the character moving on-screen, even before a gesture is given. It could also add more seamless transitions between clips played and possibly allow several clips of the same type to be played between gestures.

Based on the results discussed in Section 3, increasing the amount of samples that each model is trained on will likely improve the accuracy of the system. More testing can be done with the number of states used in the HMMs as well to find good performance. While five types of motions were selected for use in this system, there are many other types of motions available in the motion capture database. With the HMM implemented and analyzed, it would be possible to add more models to the system for gestures pertaining to other motions in the database

with little work required. Though, it is expected that recognition rates will be impacted by the addition of new gestures. Adding more observations could be added to the system involving direction in other axes to give the model additional information for more complicated motions. Doing so could also improve the accuracy of the system with the current gestures.

Acknowledgements

This project was supported by the Computing Research Association (CRA) Distributed Research Experiences for Undergraduates (DREU) summer internship program. The authors would like to thank the CRA for making this project possible.

References

[1] CMU Graphics Lab. CMU Graphics Lab Motion Capture Database. http://mocap.cs.cmu.edu/.

[2] Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. Interactive control of avatars animated with human motion data. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 491–500, 2002.

[3] Jinxiang Chai and Jessica K Hodgins. Constraint-based motion optimization using a statistical dynamic model. *ACM Transactions on Graphics (TOG)*, 26:8, 2007.

[4] Takaaki Shiratori, Moshe Mahler, Warren Trezevant, and Jessica K Hodgins. Expressing animated performances through puppeteering. 2013.

[5] Naoki Numaguchi, Atsushi Nakazawa, Takaaki Shiratori, and Jessica K Hodgins. A puppet interface for retrieval of motion capture data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 157–166. ACM, 2011.

[6] Robert Held, Ankit Gupta, Brian Curless, and Maneesh Agrawala. 3d puppetry: A kinect-based interface for 3d animation. In *Proceedings of the 25th annual ACM Symposium on User Interface Software and Technology*, pages 423–434, 2012.

[7] Tao Hongyong and Yu Youling. Finger tracking and gesture recognition with kinect. In *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*, pages 214–218, 2012.

[8] Yi Li. Hand gesture recognition using kinect. In *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*, pages 196–199, 2012.

[9] Primesense. OpenNI platform. http://www.openni.org/.

[10] Feng-Sheng Chen, Chih-Ming Fu, and Chung-Lin Huang. Hand gesture recognition using a real-time tracking method and hidden markov models. *Image and Vision Computing*, 21(8):745–758, 2003.

[11] Jonathan Glumac. Motion detection and object tracking with interactive surfaces. *Worcester Polytechnic Institute*, 2009.

[12] Mark Stamp. A revealing introduction to hidden markov models. *Department of Computer Science San Jose State University*, 2004.

[13] Maddock Meredith and S Maddock. Motion capture file formats explained. *Department of Computer Science, University of Sheffield*, 2001.

[14] cgspeed Bruce Hahne. The Daz-friendly BVH release of CMU's motion capture database. https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/daz-friendly-release.

[15] Michael Kipp. JMocap: java based viewer for motion capture data. https://code.google.com/p/jmocap/.

[16] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics (TOG)*, 21(3):473–482, 2002.