A Motion Graph Approach for Interactive 3D Animation using Low-cost Sensors

Yong Cao
Computer Science Department
Virginia Polytechnic Institute
and State University
Blacksburg, VA, 24061
Email: yongcao@vt.edu

Mithilesh Kumar
Cisco System Inc.
170 West Tasman Drive
San Jose, CA 95134 USA
Email: mithilesh@gmail.com

[1] ***Abstract*—Interactive 3D animation of human figures is very common in video games, animation studios and virtual environments. However, it is difficult to produce full body animation that looks realistic enough to be comparable to studio quality human motion data. The commercial motion capture systems are expensive and not suitable for capture in everyday environments. Real-time requirements tend to reduce quality of animation. We present a motion graph based framework to produce high quality motion sequences in real-time using a set of inertial sensor based controllers. The user's action generates signals from the controllers that provide constraints to select appropriate sequence of motions from a structured database of human motions, namely *motion graph*. Our local search algorithm utilizes noise prone and rapidly varying input sensor signals for querying a large database in real-time. The ability to waive the controllers for producing high quality animation provides a simple 3D user interface that is intuitive to use. The proposed framework is low cost and easy to setup.**

***Index Terms*—Motion Capture, 3D Animation, Accelerometer, Motion Graph.**

I. INTRODUCTION

In the past decade, motion capture technologies are rejuvenating animation industry, where high quality motion data can be captured and later edited, transformed, interpolated and stored in large motion databases (e.g., House of moves: http://www.moves.com). However, unlike video cameras, motion capture systems are still rarely used outside movie and entertainment industry. Producing high quality 3D animation requires proper planning and right tools. Therefore, it's very difficult for a regular user to generate virtual movie characters that can be shared with a large audience. There are two main reasons - high cost and the technology itself. The cost of a reasonable quality motion capture system is beyond the reach of an ordinary user. Most of the motion acquisition technology in such systems suffers from the disadvantages such as long setup time, restrict and limited capture space and occlusion (for vision-based system).

In this paper, we propose a low cost, real-time, motion estimation framework based on a small number of Nintendo© Wii^{TM}Controllers [**?**] that are easy to attach to body and impose little or no restriction on a motion capture environment. The controllers provide an accelerometer sensor based intuitive user interface for generating high quality 3D animation in real-time.

Our approach to produce the full body animation consists of three phases. During the first data collection phase, we capture the motions of a professional *performer* using a optical motion capture system. Simultaneously, we also capture 3D acceleration data from sensors attached to the performer's body. In the second phase, we build a motion graph using the approach of Kovar et al. [**?**]. The third and final phase is the motion graph search phase. We use the signals from the sensors as input signals to search through the motion graph for an appropriate sequences of clips that together resemble the action being performed by the user. Our approach performs at real-time speeds and provides interactive character control.

We evaluate our system by computing the accuracy of graph search algorithm when a user performs. We present data for motions of various actions like arm exercise, tennis strokes, golf swing and basketball. Our results indicate that high quality full body motion can be produced with good accuracy using a small number of low cost 3D acceleration sensors.

The rest of the paper is organized as follows. Section **??** presents the background, describes the related work in this area and shows the novelty of our approach. Section **??** explains the system architecture and software design that enables scalable performance. Sections **??**, **??** and **??** provide the detailed description of our approach. Section **??** shows the results and demonstrate the accuracy of our approach. We conclude the paper with a discussion of the lessons learnt and limitations. Section **??** summarizes the paper and discusses future work.

II. RELATED WORK

A. Interactive Animation Control

To avoid high system cost and long suit-up time of marker-based motion capture system, researchers seek help from other low-cost sensors for interactive animation control. Badler et al. [**?**] proposed a system that reconstructs full-body motions using four magnetic sensors and a real-time inverse-kinematic algorithm to control a standing character in some virtual environment. Another system developed by Yin and Pai [**?**] synthesizes full-body motion within one second by using a foot

[1]**Yong Cao** is the contact author. The conference name is **CGVR'10**.

pressure sensor. However, it can only generate a small range of behaviors and cannot produce motion for complex upper body movement. Chai et al. [**?**] implemented a vision based system that only requires two inexpensive video cameras. Similarly Liu et al. [**?**] use a reduced marker set for estimating human poses that is based on linear regression. However, these systems require a restrictive motion capture environment and suffer from occlusion problem of a vision based tracking system.

To combine the benefit of different motion tracking sensors, several hybrid systems were built. Their goal is to improve the quality and performance, rather than cut the system cost and suit-up time. Foxlin et. al [**?**] developed an indoor system that uses both acoustic and inertial sensors. Similarly, Bachmann [**?**] introduced an inertial-magnetic system that accounts for drifting by applying magnetic signal as reference. Most recently, Vlasic et al. [**?**] combine accelerometer sensors, inertial sensors and acoustic sensors together to capture high-fidelity motions that are comparable to the motions captured from marker based vision systems. However, the cost of the system is still high and it is not a real-time system because of the necessary post-processing time.

B. *Motion Synthesis with Motion Graph*

Motion synthesis by concatenation involves motion re-ordering and re-sequencing to produce another longer motion sequence. Motion capture data can be organized into clips, then cut and combined together again to synthesize novel motions. Kovar et al. [**?**] introduced a graph structure called *motion graph*, to model the transition points and transition edges between different motion clips. Arikan et al. [**?**], [**?**] presented a similar graph structure with motion constraints. Like with Video Textures [**?**], such re-ordering approach cannot be used for performance-driven applications, because the subtle detail from the input signal can not be represented in the result synthesized motion by re-ordering motions from the motion capture database. However, these approaches are automated techniques and very suitable for motion estimation instead of motion capture.

Lee et al. [**?**] introduce a framework for representing, searching and producing 3D animation in real-time. The graph is a hierarchical representation for enable quick search. There are 3 interfaces for generating animations - choice based, sketch based, and a performance-driven vision based interface. Except for the vision based interface, the other two are not intuitive enough for a simple user and are not very interactive. Our search algorithm shares some of the goals of this vision based interface.

In a recent work, Safonova and Hodgins [**?**] improved motion graphs but retained the key advantages - long motion sequences, a variety of behaviors and natural transitions. They synthesize motion as a linear interpolation of two time-scaled paths through a motion graph. Interpolation provides ability to synthesize more accurate and natural motion as physically realistic variations. However, their approach is computation intensive and is therefore not suitable for real-time synthesis.

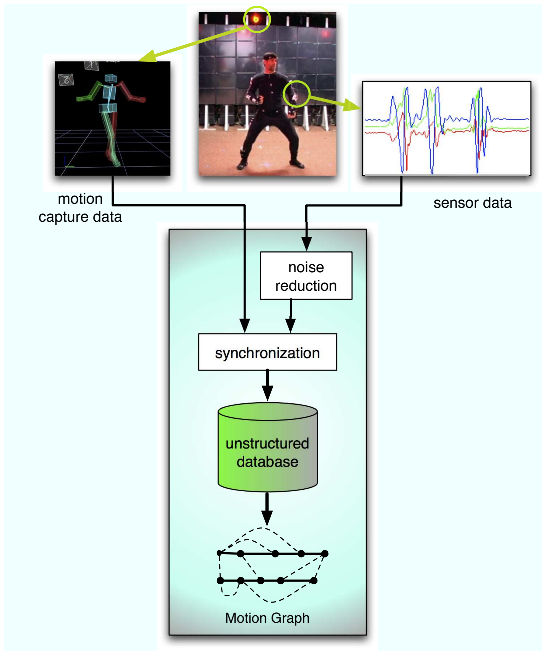

Fig. 1. Data collection using Vicon motion capture system and Nintendo Wii Controllers. Ultimately, all collected data is represented as a *Motion Graph*

Xie et al. [**?**] use low-cost sensors, similar to ours, to develop a lo-cost, data-driven framework for motion estimation. This work provides the foundation for our work, since they succeed in generating high-quality data based on a local linear model learned from a high quality motion database. The linear model is an interpolation model based on RBF. Using this model and the control signals from the sensors, it is now possible to synthesize new poses. One limitation of this work is that it is an offline strategy and the synthesized result is not smooth.

III. System Overview

We present a real-time motion estimation framework that tracks the user's body movement and produces a motion that closely resembles the users action. In contrast to motion capture systems, we do not attempt to capture the subtle details of the user's action. Our data-driven approach relies on a structured database of motion capture data and a set of sensors that are used to track body movement. Our approach consists of three phases - Data collection, data representation and real-time motion generation.

A. *Data Collection and Representation*

Figure **??** illustrates these two steps. We perform a series of off-line motion capture sessions in the studio using an optical motion capture system for animation data and a set of inertial sensors for acceleration data. The sensor data is pre-processed to reduce noise. Following this, we synchronize the motion data with the sensor data in order to get precise frame-to-frame mapping. All the data is then stored in a database and then converted into a structured database called *motion graph*.

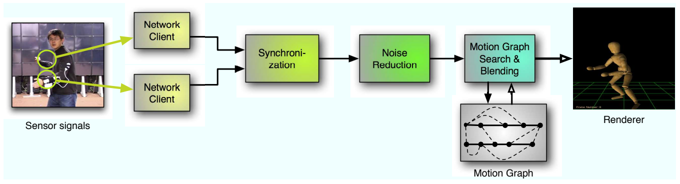

Fig. 2. System architecture for real-time motion generation

A motion graph is a directed graph that connects a motion capture frame to all other similar frames such that a transition is possible while maintaining the continuity of motion. The motion graph is structured and efficient representation and allows real-time search for motion clips that satisfy a set of user supplied constraints.

B. Motion Generation

The data collection and representation are carried out only once with the final result being a motion graph. The motion graph then is used multiple times to conveniently produce different kinds of animation in everyday surroundings. Our wearable motion estimation system consists of two ordinary Wii^{TM}controllers that transmit signals to a bluetooth capable terminal. During motion generation, these input sensor signals coming from different sensors are synchronized in time to each other and any noise is removed. We then use the input signals to query the motion graph for walks in the graph that match to the input signals. The walk consists of motion clips that are stitched together and blended at the transition points to create a continuos motion sequence. The motion estimation system architecture is shown in Figure **??**.

IV. Data Collection

During the studio motion capture session, we capture 2 types of data. The first is a high quality motion capture data, captured using Vicon optical motion capture system. Synchronously, we capture acceleration data using accelerometer sensors attached to the limbs of the performer. Only one motion capture session was required in which we captured a total of 86 seconds of data, including arm exercising, tennis, basketball and golf motions. For building the database used for the examples presented in this paper, we had only one subject perform all the actions.

For optical motion capture, we use a Vicon system with 8 Vicon MX series cameras for high quality motion capture at 60 Hz. The motion sensors are 3D accelerometers (Wii^{TM}Nintendo controllers) with a range of $\pm 3g$ and built-in Bluetooth®interface for data transmission at a peak rate of 100 Hz. The interface based on these wireless inertial sensors is cheap, easy to set-up and unlike vision based system, does not suffer from occlusion. There are 45 retro-reflective markers and eight sensors attached to the performer's body. The sensors are attached to the arms and legs since they provide most of the movements for a majority of human actions. We only used data from the 2 sensors attached to each forearm. Actions present in our database mostly employ movement of the arms and therefore the 2 sensors provided sufficient constraints for the search algorithm. For complex actions involving legs, we would be required to use more sensors. Each sensor transmits its 3D acceleration data to the data collection computer where the data is passed through a noise reduction filter. Next, it is converted into sensor frames, synchronized with motion data frames and stored into the database.

The created database with N frames of data is a collection of values of the form $(\mathbf{c}_t, \mathbf{m}_t)|t = 1, \ldots, N$. $\mathbf{c}_t$ is a frame of sensor data with 6 dimensions and it represents the 3D acceleration measures of four or eight sensors on the body at time t. $\mathbf{m}_t$ is a frame of optical motion capture data and it represents a pose at time t. The pose is represented in the form of local joint rotation in the quaternion format.

V. Data Representation

Given a database of motion capture data, we create a directed graph called a *motion graph*. The idea of motion graph is not new and the structure of our motion graph that we use is an extension of the data structure used by Kovar et al. [**?**]. There are two kind of edges in the graph. As seen in Figure **??**(a), the solid edge represents a piece of original motion capture data which we will refer to as *clip*. The dashed edge represents a *transition* from one end of a clip to the beginning of another. Thus the nodes are junctions called *transition points* where transitions begin and end. Transition points are also the place where a clip starts or ends. A local loop is a transition to self node. Self loop has been introduced for uniformity and is required to transition to the outgoing clip edge from current node. In Figure **??**, node 7 is a dead-end node because there is no outgoing edge from this node. The motion graph is built only once and stored in a text file. In every subsequent run, of the application, the motion graph is loaded from this file at run time.

Pruning the Motion Graph Once transitions are between clips have been identified, we should remove unwanted nodes and edges that do not contribute positively to the search algorithm. Pruning the motion graph for unwanted transitions is required for the following reasons: (1) remove acyclic regions in the motion graph; (2) high quality of transitions; (3) reduce complexity and compress the motion graph.

Merging Similar Transitions We apply convolution to obtain the local minima for selecting transitions. However, there will be transitions that have source frames which are adjacent in the motion sequence and very identical to each other while the destination frames are scattered. We must detect these nodes and represent them with one single node while keeping all the transitions. Our compression technique retains the functionality of the motion graph with the help of a more compressed and efficient transition.

As shown in Section **??**, this scheme will be beneficial in the search phase of the motion graph. Similar to scenario described above, we will have another scenario in which the source nodes for transitions are not in a neighborhood, but the destination nodes lie in the same neighborhood. Although

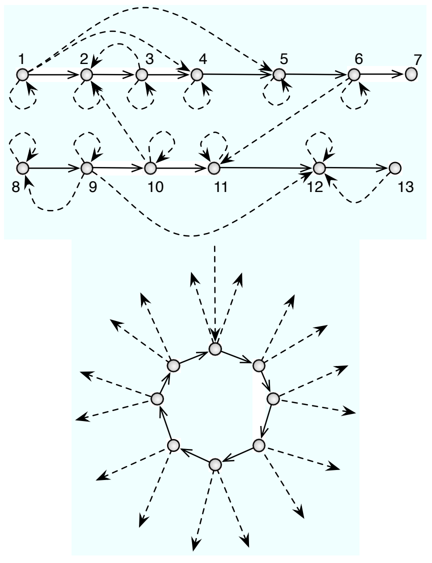

Fig. 3. (a) An example motion graph generated from two sets of motion capture data. (b) The default motion sub-graph.

merging these nodes have no effect as far as the search algorithm is concerned, it will help in making the motion graph more compact, thus reducing memory consumption and boosting performance.

VI. Searching the motion graph

In the previous section we built a motion graph from the motion capture database. In this section, we describe the process of generating motion sequences by searching the motion graph for motions corresponding to actions being performed by the user. The user wears two sensor controllers in each of his arms. The total setup time is less than 2 minutes. Since our system does not suffer from occlusion, the user can position himself in any direction, but should be close enough to the terminal to maintain Bluetooth connectivity. For examples in this work, the user performs several actions that mostly require movement of the upper body. For this reason, it is sufficient to use only 2 controllers.

At this stage there are 2 tasks to accomplish. First we must search in the motion graph for paths that satisfy user constraints in real-time. Secondly the resulting graph walk needs to be converted to continuous motion.

A. Local online search

Given a current node, our novel graph search algorithm uses the input sensor signals as the query for searching a matching transition from the node. Our approach is local because we only search for transitions possible from the current node. Secondly, our approach is online because search and rendering is performed at real-time rates. Unlike the graph building phase, in the search phase we will use only the motion sensor frames. However, every motion capture frame of a clip edge in the motion graph has a corresponding motion sensor frame.

At any given time t, we maintain a sliding window buffer, $\mathbf{U}_t$ of size CSW for the input sensor signals. If the current node is n_t, we obtain all the transitions possible from n_t. Then we find the minimum of the distance, d_{min} as shown in Equation**??**, between the input signal and each of these transitions as the weighted square of Euclidian distance.

$$d_{min} = \min\left(\sum_{i=0}^{CSW} w_i ||cs_i - cs'_i||^2\right) \tag{1}$$

Here cs_i is an input sensor signal frame in the buffer $\mathbf{U}_t$ and cs'_i is a sensor signal frame from the motion graph corresponding to a transition from n_t with cs_0 being the first frame in the destination clip edge. The weight w_i is a linear weight given by $w_i = \frac{1}{CSW}$.

Let us assume that the first and the last frame of the current clip edge is $\mathbf{A}_{i-l}$ and $\mathbf{A}_i$ respectively where l is the length of the clip edge. The process for finding d_{min} begins when $\mathbf{A}_{i-l}$ is being rendered and is repeated after every subsequent frame until we reach frame $\mathbf{A}_{i-\frac{k}{2}}$. The choice for this frame is based on our blending technique. After every iteration we update d_{min} if a lower value is found. When we reach the frame $\mathbf{A}_{i-\frac{k}{2}}$, we stop searching and proceed with next step - to select an appropriate transition based on our search results. At this state our search algorithm has yielded d_{min}, which is the transition to the clip which is closest to the query signal.

If d_{min} is below an acceptable distance threshold, we select the transition corresponding to it and proceed with blending. However, it would be easy to encounter situations in which d_{min} is above the distance threshold and the transition is acceptable. This would typically happen if the user is performing an action that does not exist in the database. It would also happen if the user did not perform an action well enough to get recognized. In such cases we select a transition to a node in a special sub-graph called *default motion sub-graph*. However, not all nodes will have a transition to the default sub-graph. In this case we choose a self-transition and continue playing frames from the new clip until we encounter a transition to the default motion sub-graph.

A default motion sub-graph is our extension to a pure motion graph based approach. The default motion sub-graph, as shown in Figure **??**(b), is constructed from a specially selected motion capture data in which the skeleton pose is in a neutral position in all the frames.

Once the default poses are decided, we take 0.5 second (30 frames) of continuos motion from a clip such that all the frames are very similar to each other. This motion sequence, called the *default motion sequence*, can be easily found by running though the entire motion capture database and finding a 30 frame sub-sequence of frames, all similar to the given default pose. Using these frames, we construct a small and special motion graph called the *default sub-motion graph* which has as many nodes as number of frames in the default motion sequence. The end node is connected to the beginning node, forming a circle of continuous motion. There is only one gateway node for all incoming transitions. However, all nodes have outgoing transitions. We now connect this sub-graph to the rest of the motion graph by creating transitions from all clip edges whose end frame is similar to the default frame. Finally, it should be possible for us to transition from any node in the default motion sub-graph to the selected nodes of the motion graph without waiting for the entire default clip to finish playing. For this we duplicate all the transitions that were created for the first node of the default motion sub-graph, to the selected nodes of the motion graph.

The default motion sub-graph is a unique feature required to play animation whenever the search algorithm returns a poor transition. It is also used to play a clip that indicates that the system is waiting for user to perform an action.

B. Generating motion

A walk through the motion graph results in a motion being generated. After the search phase, we have identified a path consisting of clip and transition edges in the motion graph that should be converted into continuous and smooth motion. Since we are stitching clips together, we must first align the clips such that the start frame of new clip matches the global translation and orientation of the previous clip played. Then, we blend the motion at the junction to produce smooth transition.

VII. Results

Our system is capable of capturing motions that are similar to the ones in the motion database. The user has two Wii$^{\text{TM}}$controllers attached to his arms and connected to the terminal via Bluetooth®. We asked the user to perform tennis forehand, tennis serve, basketball dribble, basketball shot, arms exercise and golf swing. The actions were performed in random order. A sequence of skeleton poses along the corresponding images from video is shown in Figure **??**. The quality and accuracy of our approach is shown in the accompanying video. The examples presented in this paper were run on a terminal with using 2.33 GHz Intel Core 2 Duo processor and 2GB of memory. The real-time frame rate achieved 60 frames per second.

For evaluating our system, we follow a two-step procedure. In the first step we measure the accuracy of our search algorithm to correctly match user actions with motions in the database. In the second step, we measure the stability of our system.

Accuracy The accuracy measure reflects the ability of our search algorithm to find best match for user actions. If the user performed an action N times, we measure the number of times the action was *correctly recognized* (CR), number of times the action was *incorrectly recognized* (IR) and number of times the action was *not recognized* (NR). An action is considered not recognized if the avatar does not respond to the user action and instead maintains the default pose. Each of these measurements are summarized in Table **??** for $N = 50$, the number of repetitions of each action. The overall accuracy achieved was 91%.

TABLE I
Accuracy of the online, local search algorithm

Action	$\frac{CR}{N}$	$\frac{IR}{N}$	$\frac{NR}{N}$
Right arm dumbell	0.88	0.00	0.12
Left arm dumbell	0.84	0.0	0.16
Right arm lift	0.96	0.0	0.04
Left arm lift	0.96	0.04	0.0
Tennis forehand	0.82	0.0	0.18
Basketball dribble	0.88	0.06	0.06
Basketball shot	0.76	0.0	0.24
Golf	0.94	0.0	0.06

Stability The stability test tells us if our system is intelligent enough to recognize intervals when the user is not performing any action. During such cases, it is expected that an appropriate default clip is being played giving the impression that the avatar is waiting for the next instruction from user. An error condition occurs if the system wrongly plays a motion sequence, instead of remaining in default state. We call this a *Type I* error. For measuring the Type I error, we asked the user to stand steadily in each of the three defaults poses.

In another case, if the user performs arbitrary actions (not present in database), then the search algorithm should not return anything and a default clip should be played. The action to be performed for this test is chosen carefully so that it is not similar to any of the actions in the database. Again, an error condition is produced when the system wrongly plays a motion sequence, instead of a default pose clip. We call such errors as *Type II* errors. For this test, the user was asked to perform two actions. First we asked the user to move right arm in circular fashion, as if drawing a circle on a blackboard. The second action was to move both arms as if the user is running.

All actions were performed continuously for 30 seconds and each action was repeated 10 times. The results are presented in Table **??**. We see that while the user is not moving, there are no errors, but when the user is moving arbitrarily, the system is occasionally confused and selects a wrong transition. However, the low percentage in error for Type II errors suggests that our system is stable during such conditions.

TABLE II
Stability test

Error Type	**Action**	**Error (%)**
Type I	Default pose 1	0
Type I	Default pose 2	0
Type I	Default pose 3	0
Type II	Running	0
Type II	Right arm circle	10
Type II	Dumbbell, both arms	0
Type II	Arm lift, both arms	0

VIII. Conclusions

We present a framework for estimating full body motion in real-time using a small set of low cost inertial sensors. Our three step approach involves data collection, building a motion graph and motion generation though local graph search. Data collection is performed in the studio and produces a database of time synchronized high quality motion capture data and sensor data. Using this database, we then construct a data structure for 3D animations called a motion graph. Creating an efficient motion graph is not trivial, since it involves finding the best transition points from a pool of candidate transitions. We prune the graph to remove redundant transitions and dead-ends and introduce a new compression technique to improve search performance.

Using this motion graph, we can generate various new motion sequences by concatenation of clips obtained from the motion graph search. In the search phase, we proposed a local online algorithm that uses the sensor signals as a query to the motion graph and returns an appropriate transition corresponding to the user's current action. When the search algorithm is unable to understand the user's action we play the default motion clip until the user performs an action that can be recognized. We extended the commonly used motion graph technique to introduce a default motion sub-graph. A walk through this circular sub-graph produces the default motion clip.

The results obtained show the effectiveness of of our framework. We achieved accuracy of 91% while maintaining a lag of 1.33 seconds. The quality of the generated motion sequence is same as original motion capture data, since we follow a cut and paste strategy and synthesize frames only during transitions between different clips.

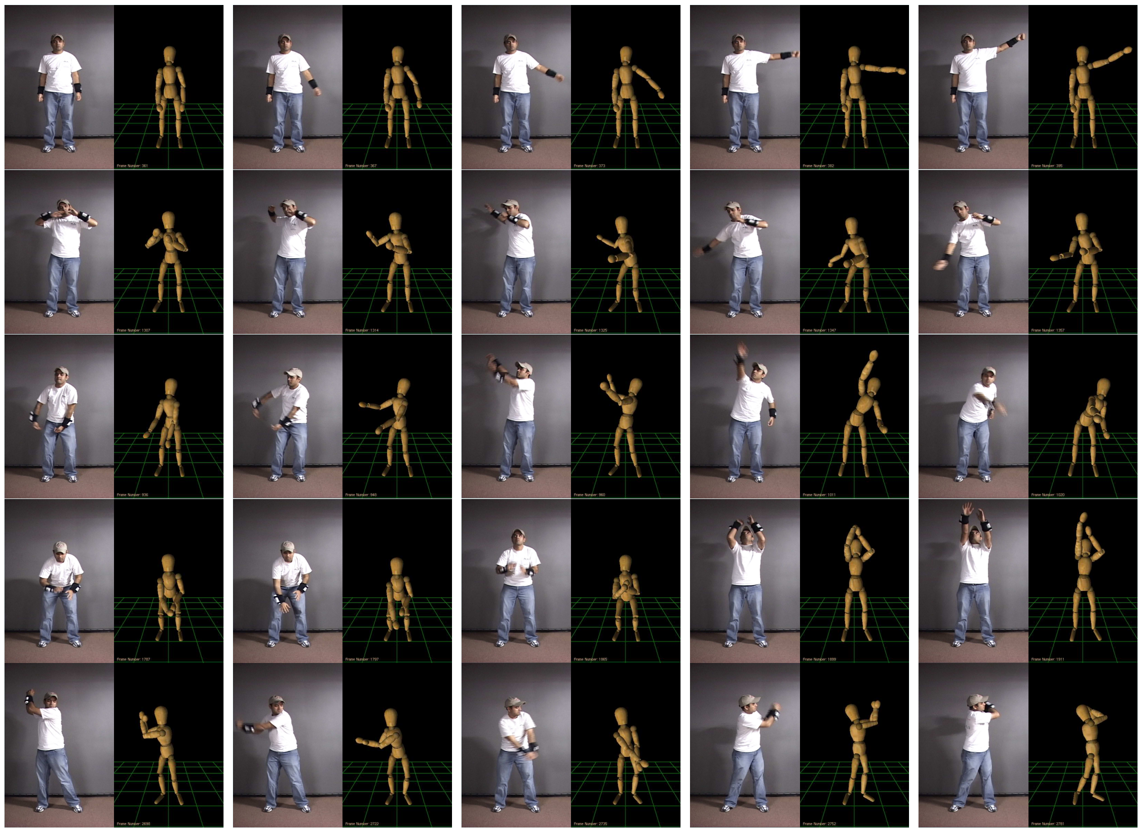

Fig. 4. Five different actions (one in each row) generated by our system. Each frame shows on the left side the actual pose and on the right side the a pose from the generate motion sequence. For the purpose of comparison, we have presented the results after removing the lag. *(Image by author)*