Digital Signal Processing 23 (2013) 1441–1450

Contents lists available at SciVerse ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

Motion capture and human pose reconstruction from a single-view video sequence

Uğur Güdükbay *, İbrahim Demir, Yiğithan Dedeoğlu

Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey

ARTICLE INFO

Article history:
Available online 19 July 2013

Keywords:
Motion capture
Human pose reconstruction
Single camera
Uncalibrated camera
Computer vision
Animation

ABSTRACT

We propose a framework to reconstruct the 3D pose of a human for animation from a sequence of single-view video frames. The framework for pose construction starts with background estimation and the performer's silhouette is extracted using image subtraction for each frame. Then the body silhouettes are automatically labeled using a model-based approach. Finally, the 3D pose is constructed from the labeled human silhouette by assuming orthographic projection. The proposed approach does not require camera calibration. It assumes that the input video has a static background, it has no significant perspective effects, and the performer is in an upright position. The proposed approach requires minimal user interaction.

1. Introduction

Human pose reconstruction is a significant research problem since it can be used in various applications. Motion capture and motion synthesis are expensive and time consuming tasks for articulated figures, such as humans. Human pose estimation based on computer vision principles is an inexpensive and widely applicable approach. In computer vision literature, the term human motion capture is usually used in connection with large-scale body analysis ignoring the fingers, hands and the facial muscles, which is also the case in our work. The motion capture problem we try to solve can be defined as follows: given a single stream of video frames of a performer, compute a 3D skeletal representation of the motion of sufficient quality to be useful for animation. The animation generation is an application of motion capture where the required accuracy is not as high as in some other applications, such as medicine [1].

In this study, we propose a model-based framework to reconstruct the 3D pose of a human for animation from a sequence of video frames obtained from a single view. The proposed framework for pose reconstruction starts with background estimation. Once the background is estimated, the body silhouette is extracted for each frame using image subtraction. Then, 2D body segments are automatically labeled on the human body silhouette using a model-based approach. Finally, the 3D pose is constructed from the labeled human silhouette by assuming orthographic projection. The approach proposed in this work does not require camera calibration and it uses a video sequence obtained from a single camera. The proposed framework assumes that the input video has a static background, it has no significant perspective effects, and the performer is in an upright position. Our approach computes joint locations automatically for the unoccluded parts and requires minimal user interaction for the specification of foreshortening directions, the orientation of the body, and joint locations in the highly occluded parts. We tested the proposed framework on various video sequences and obtained reasonable reconstructions of the human figure motion.

The proposed framework can help the professional animators produce an initial version of the motion using real motion data capture from a single-view video sequence, which then can be refined further. It can also enable animators to use motion capture data to construct motion libraries from public resources.

The remainder of this paper is organized as follows. Section 2 gives an overview of the problem and the related work. In the next section we propose our framework for human motion capture from a video sequence obtained from a single camera and animation using the captured motion data. Experimental results are discussed in Section 4 and finally we conclude the paper with Section 5.

2. Overview and related work

There are two main motion control techniques for animating articulated figures: kinematics and dynamics. Kinematics methods use time-based joint rotation values to control the animation while dynamics methods use force-based simulation of movements [2]. Creating data for these techniques can be done manually by talented animators or can be captured automatically by different types of devices [3,4].

* Corresponding author. Fax: +90 312 266 4047.
E-mail addresses: gudukbay@cs.bilkent.edu.tr (U. Güdükbay), ibrahimdemir.mail@gmail.com (İ. Demir), yigithan@cs.bilkent.edu.tr (Y. Dedeoğlu).

1051-2004/$ – see front matter
http://dx.doi.org/10.1016/j.dsp.2013.06.008

Motion capture is an effective way of creating the motion data for an animation. It provides realistic motion parameters and permits an actor and a director to work together to create a desired pose, which may be difficult to describe specifically enough to have an animator recreate manually [1]. There are various application areas for motion capture techniques such as virtual reality, smart surveillance systems, advanced user interfaces and motion analysis and synthesis [5,6].

Producing high-quality animation is a challenging task. Capture systems that overcome these challenges have previously required expensive specialized equipment. Computer vision techniques can be used as a replacement to obtain animation data in an easier and cheaper way. Human motion capture systems generate data that represents measured human movement, based on different technologies. According to the technology used, currently available human motion capture systems can be classified as either non-vision based, vision based with markers, or vision based without markers. Ideally, the capture of motion data should be inexpensive and easily available. Using a single standard video camera is an attractive way of providing these features. It offers the lowest cost, simplified setup, and the potential use of legacy sources such as films [1]. The method proposed in this paper is vision based and does not require additional equipment, such as markers. Because a camera can have a resolution measured in megapixels, such vision-based techniques require intensive computational power [7].

As a commonly used framework, 2D motion tracking is only concerned with the human movement in an image plane, although sometimes people intend to project a 3D structure into its image plane for processing purposes. This approach can be cataloged with and without explicit shape models [8]. The creation of motion capture data from a single video stream seems like a plausible idea. People are able to watch a video and understand the motion, but clearly, computing human motion parameters from a single-view video sequence is a challenging task [1].

Felzenszwalb and Huttenlocher present a framework for part-based modeling and recognition of objects in [9]. In one of their experiments they try to find articulated human body in images. Their approach use pictorial structures to find instances of objects in images and requires manual labeling of joints for learning. This framework is not restricted to human body and can be used for other articulated objects.

Ramanan and Forsyth describe a system that can annotate a video sequence in [10]. Their system works by tracking people in 2D. The success of their tracking algorithm depends on segmentation and clothing of the subject.

Rosales et al. introduce a framework for 3D articulated pose recovery, given multiple uncalibrated views in [11]. A statistical inference method known as Specialized Mapping Architecture (SMA) is used. 2D joint locations across frames and views are provided by the SMA. The SMA requires training data to be able to provide 2D joint locations. The approach used in this paper requires multiple camera input.

Mori and Malik present a method to estimate the human body configuration in images [12]. In their work, 2D joint locations are found automatically by shape context matching technique. 2D exemplar views are stored and locations of joints are manually marked and labeled in these views to be used for future. The tested image is matched to the stored views using the shape context matching technique. The technique is based on representing a shape by a set of points from internal or external contours, found using an edge detector. If sufficient similarity is shown by shape context matching technique to a stored view, the stored joint points are transferred to the test image. Given the joint location, the 3D body configuration and pose are estimated using the Taylor's algorithm. Since their approach requires user intervention for manual labeling, it is difficult to use for videos.

Kakadiaris and Metaxas [13] describe a method for 3D model-based tracking and shape estimation of human body parts using images from multiview cameras in orthogonal configurations. They later extended their approach for estimation of human movement from multiview video sequences and create animation sequences accordingly. Their approach works well for the case of tracking upper body extremities [14].

Shakhnarovich et al. present an algorithm that learns a set of hash functions that efficiently index examples in a way relevant to a particular estimation task in [15]. Their algorithm requires training and is tested for human upper body images with synthetically created data. Their method fails on some samples but for the most of the time it gives accurate results. Their approach is fast for learning phase when compared to similar methods but their experiment does not entirely cover 3D pose recovery.

Ren et al. present a method that uses example based approach for finding 3D body configuration in [16]. Multiple cameras are used in their approach. Domain specific database is created for three views and tested images are matched to the stored images. The matching process uses discriminative local features that are computed on the silhouette images.

Bregler and Malik [17] describe a vision-based motion capture technique that can recover articulated human body configurations in complex single-view video sequences. They use the product of exponential maps and twist motions for visual tracking, and integrate it for differential motion estimation. Ye et al. [18] describe an accurate 3D human pose estimation method for depth map sequences obtained using Microsoft Kinect® (Kinect is a registered trademark of Microsoft Co.). Wei and Chai [19] describe a video-based motion modeling technique for generating physically realistic human motion from uncalibrated monocular video sequences. They employ physics, contact constraints, and 2D image measurements to reconstruct human body poses and other physical quantities such as torques and forces, to reconstruct physically realistic motions. Vondrak et al. [20] propose a motion human motion estimation method for monocular videos employing a biped controller with a balance feedback mechanism for encoding motion control as a sequence of simple control tasks. They demonstrate their approach for walking, jumping, and gymnastics.

There are notable studies for markerless motion capture using multiple view video sequences. Ballan and Cortelazzo [21] propose a markerless motion capture of skinned models using optical flow and silhouette information using a four camera setup. They transfer the reconstructed motion to a synthetic human model and animate it using the reconstructed motion. Gall et al. [22] propose a method for capturing the performance of a human or an animal from a multi-view video sequence. Michoud et al. [23] propose an extended shape-from-silhouette algorithm to reconstruct the motion of a human model in real-time from three calibrated Webcams. Gorelick et al. [24] represent actions as space–time shapes and detect actions of human figures accurately in single-view walking and dancing sequences. Ofli et al. [25] describe a framework for unsupervised video analysis of dance performances from multiview video recordings and animate a virtual 3D character accordingly. They model each dance figure using a set of Hidden Markov Models.

Our approach for constructing the 3D pose is close to the approach defined in [26], but in our approach the necessary user interaction is significantly reduced. In order to construct the 3D pose, the joint coordinates of the human figure are needed. Unlike Taylor's approach, which gets the joint correspondences from the user, our approach computes these points automatically. In Taylor's work, the motions of multiple human figures can be captured because the joint correspondences for all of them can be specified manually whereas our framework works for a single human figure in video.

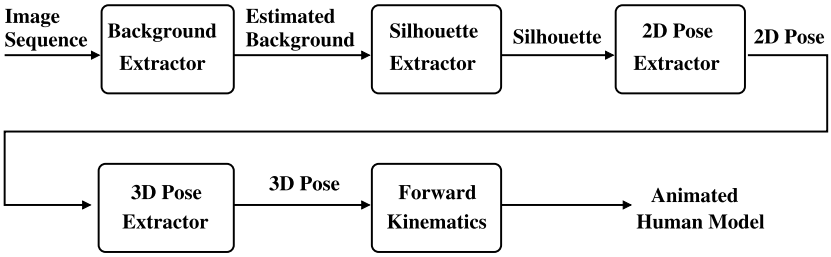

Fig. 1. The proposed framework for human motion capture from a single-view video sequence and animation using the motion capture data.

3. Motion capture and animation system

Capturing human motion parameters has various challenging problems. Thus, the solution for such a problem requires various methods and algorithms. In this section we give an overview of our framework and the details of each stage.

The proposed framework finds the posture of a human using the joint correspondences calculated automatically from the image and the depth direction of each body segment, which is provided by the user using the mouse. In [26], the joint correspondences are provided by the user in addition to the depth direction. Our aim is to use image processing, computer vision algorithms, and computer graphics techniques to extract human motion parameters (namely body postures) with minimal user effort. After the joint angles are calculated, the remaining information needed to construct a 3D pose is the depth information (inward or outward) of each body part, which rarely changes during a video sequence. The user only has to interact whenever a body segment changes its depth direction.

The proposed framework for human motion capture and animation is shown in Fig. 1. For capturing the 3D pose from the given image sequence, the first stage estimates the background of the video scene. Then, we find the silhouette of the performer by subtracting the estimated background from each frame. The extracted silhouette of the performer is given as input to the 2D pose extractor. The 2D pose extractor finds the joint coordinates of the performer on the image by using a model-based approach. A 2D stick human model (see Fig. 2) is fitted onto the silhouette. In this way, the joint coordinates of the performer are matched to the joint coordinates of the 2D stick human model. The joint coordinates of the 2D stick model will be given as input to the 3D pose estimator. The 3D pose estimator computes the 3D joint configurations from the 2D joint coordinates and the depth information of the human model for each body part. Finally, the 3D human model is animated based on the 3D joint configurations computed from the video sequence [27,28]. We use forward kinematics to generate the human posture according to the calculated joint angles. Forward kinematics calculates the positions and orientations of the limbs and the end effectors of the human model starting from the root of the joint hierarchy towards the end effectors (cf. Fig. 2).

3.1. Human model

The skeleton can be represented as a collection of simple rigid objects connected by joints, which are arranged in a hierarchical manner. These models, called articulated bodies, can have various degrees of articulation. The number and the hierarchy of joints and limbs and the degrees of freedom (DOF) of the joints determine the complexity of the model. The DOF of a joint is the independent position variable that is necessary to specify the state of a joint. The joints can rotate in one, two, or three orthogonal directions. The number of orthogonal directions determines the DOF of a joint. A human skeleton may have many DOFs. However, as the number of DOFs increases, the methodology used for controlling the joints becomes more complex.

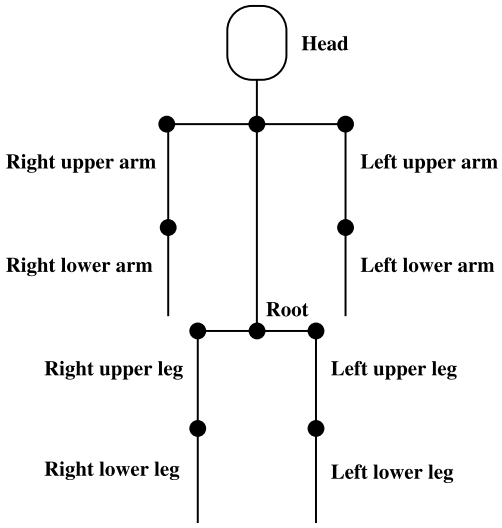

Fig. 2. The human body model used in the implementation.

For the sake of our model-based framework, we have to think about tradeoff between the accuracy of the representation and the number of parameters for the model that need to be estimated. In our work, we are interested in large human body movements. Hands or facial expressions are not considered. To reduce the computational complexity of the model we use a simple 3D articulated human model to capture the motion. Our articulated human model consists of 10 cylindrical parts representing head, torso, right upper leg, right lower leg, left upper leg, left lower leg, right upper arm, right lower arm, left upper arm, and left lower arm (see Fig. 2). Each cylindrical part has two parameters: *radius* and *length*. For each cylindrical part there are up to three rotation parameters: θ_X, θ_Y, and θ_Z. In total, there are 23 DOFs for the human model: 3 DOFs for the global positioning of the human body, 1 DOF for the head, 3 DOFs for the torso, 2 DOFs for the right upper leg, 2 DOFs for the right lower leg, 2 DOFs for the left upper leg, 2 DOFs for the left lower leg, 2 DOFs for the right upper arm, 2 DOFs for the right lower arm, 2 DOFs for the left upper arm, 2 DOFs for the left lower arm.

3.2. Background estimation and silhouette extraction

We need to estimate the background of the scene to extract the silhouette of the performer from the video sequence. We make use of the background subtraction method described by Collins et al. [29]. We analyze a few frames to find the regions that do not change. The silhouette is the main feature extracted from the video frames and used in our framework. Our 2D pose extractor

Table 1
Body part ratios and joint limits: (a) the ratios of the lengths of different parts to the length of the human model; (b) the ratios of the radii of different parts to the length of the human model; (c) the joint limits in the human model.

(a)		(b)		(c)	
Body part	Ratio	Body part	Ratio	Joint	Start/End limit (degrees)
Height	175/175	Head	20/175	Neck	45/135
Head	25/175	Torso	40/175	Waist	45/135
Torso	52/175	Upper arm	10/175	Right hip	240/300
Upper arm	25/175	Lower arm	10/175	Left hip	240/300
Lower arm	35/175	Upper leg	20/175		
Upper leg	46/175	Lower leg	20/175		
Lower leg	52/175				

takes the silhouette of the performer as input and processes the silhouette to find 2D joint coordinates of the performer.

We detect the silhouette on a frame as the difference between the frame and the background. We apply the threshold value to decide whether a pixel belongs to the silhouette or the background. If the absolute value of the difference between the pixel of the frame and the background pixel is greater than the specified threshold, then the pixel is taken as a silhouette pixel, otherwise the pixel is regarded as a background pixel. Our method of estimating the background may not be suitable in video clips where there are dark shadows or too much visual noise.

3.3. 2D pose extraction

The aim of 2D pose extraction is to find the joint coordinates of the performer by using the human silhouette. We extract joint coordinates of a human actor by using a model-based technique. We model the human as an assembly of cylinders. We match the silhouette with the human model iteratively. After fitting the human model onto the silhouette, we can use the joint coordinates of the human model as the joint coordinates of the performer. In our method the performer is assumed to be in an upright position.

During 2D pose extraction, we do not know whether a leg is a left leg (arm) or right leg (arm). When we talk about 2D pose extraction we use "left leg" to mean the leg that is on the left of the image. The distinction between left and right is actually done during 3D pose estimation. For 2D pose extraction we have to be sure that the segments on the left of the image are identified as left and the segments on the right of the image are identified as right.

The process of finding the 2D pose starts by detecting the torso location. We locate the y coordinate of the torso from the relative ratios shown in Table 1. We take the horizontal middle point of the silhouette as the x coordinate of the torso. Then we find how much the torso is rotated around the normal axis of the image plane using Algorithm 1, as shown in Fig. 3(a). Then we detect the head rotation using Algorithm 1. We analyze the contour of the silhouette using Algorithm 2 to find how much the lower left leg angle, lower right leg angle, upper left arm angle, upper right arm angle, lower left arm angle, and lower right arm angle are rotated around the normal axis of the image plane. Then, we find the rotation angle of the upper left leg and the upper right leg by Algorithm 1, as shown in Figs. 3(b) and 3(c). After finding each rotation angle we find the foreshortening of each segment by using Algorithm 6. The methods for finding the foreshortening of the upper arm and the torso are shown in Figs. 3(e) and 3(d), respectively.

3.3.1. Finding orientation of body parts

Algorithm 1 determines how much one or more body segments is rotated around the normal axis of the image. The technique used in Algorithm 1 is similar to the method used in [30]. Before going further, we have to define the similarity between two images.

We measure similarity between two images I_1 and I_2 by an operator $S(I_1, I_2)$ as described in [30]. The similarity operator only considers the area difference between the two shapes; i.e., the ratio of the positive error p (representing the ratio of the number of pixels in the silhouette but not in the human model to the total number of pixels of the human model and the silhouette) and the negative error n (representing the ratio of the number of pixels in the human model but not in the silhouette to the total number of pixels of the human model and the silhouette) which are calculated as

$$p = \frac{(I_1 \cap I_2^C)}{(I_1 \cup I_2)}, \tag{1}$$

$$n = \frac{(I_2 \cap I_1^C)}{(I_1 \cup I_2)}, \tag{2}$$

where I^C denotes the complement of I. The similarity between the two shapes I_1 and I_2 is calculated as

$$S(I_1, I_2) = e^{-p-n}(1-p). \tag{3}$$

Algorithm 1 takes the fragment of the human silhouette that contains the body segments to be searched as the input. The algorithm also takes the lower and upper joint angle limits. The lower and upper joint angle limits used in our implementation are listed in Table 1. We divide the angle range specified by the lower and upper limits to a number of intervals and we measure how well the silhouette is covered by the segments for each interval. We find the division (angle) that covers the silhouette best. At each iteration we narrow the search interval by selecting the next search interval for the joint angle as the division that gives best covering result merged with its neighbor divisions since there is a possibility the best angle can be in these divisions. We recursively continue to narrow the search interval and when the search interval is smaller than a threshold value the algorithm returns the angle that corresponds to the best fit. Examples of the application of this algorithm for the torso, the upper left leg, and the upper right leg are shown in Figs. 3(a), 3(b), and 3(c), respectively.

3.3.2. Contour analysis

This method is used to find how much the segments of the body (lower left leg, upper right leg, upper left arm, lower left arm, upper right arm, and lower right arm) are rotated around the normal axis of the image plane. We use contour analysis to find angles. We do not use Algorithm 1 for the arm segments because of the high possibility of occlusion with other parts. For lower legs, we cannot use Algorithm 1 because we have to first find the upper legs. This is because we do not know the end points of the upper legs. Trying to find the upper leg angle by using Algorithm 1 is also not appropriate. Since upper legs might be occluded by clothes or

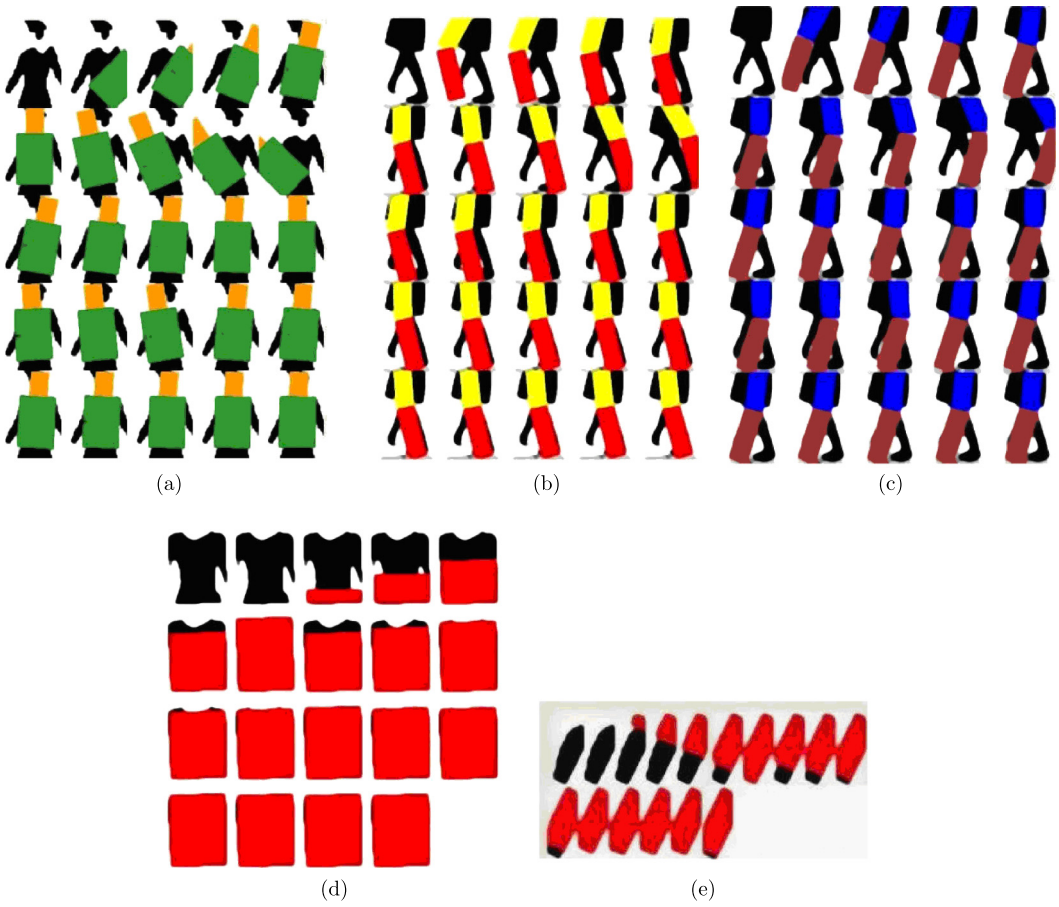

Fig. 3. Iterative estimation: (a) the torso orientation; (b) the upper left leg orientation; (c) the upper right leg orientation; (d) the foreshortening ratio of the torso; (e) the foreshortening ratio of the upper arm.

adjacent to each other, it is difficult to find their joint angles. For this reason, we use Algorithm 2, which is based on contour analysis, for the lower legs. After finding the angle of the lower legs by using Algorithm 2 we find upper leg angles by using Algorithm 1. While finding the upper leg angle with Algorithm 1, we combine the upper leg and the lower leg into a single unit, as shown in Figs. 3(b) and 3(c). For some segments, Algorithm 1 gives better results if the segment is combined with other segments. For example, while we are determining the torso angle, the torso and the head are also taken together as shown in Fig. 3(a). The head helps to find the torso angle easily. By the same logic, since the lower leg angles are found with the contour analysis method, knowing the lower leg angle helps to find the upper leg angles easily, as shown in Figs. 3(b) and 3(c).

The contours of the left arm, right arm and lower left leg and lower right leg are extracted from the silhouette by traversing horizontal scan lines over the silhouette. A curve is extracted for each part for this purpose. For the lower left leg and the lower right leg, we compute the joint angle from the curves, as shown in Algorithm 2.

For the arms, we have to find two angles: one for the lower arm and one for the upper arm. We extract two angles from the arm curves by trying to find potential elbow corners on their point lists. We use Algorithm 4 to find the elbows. Algorithm 4 tries some points as if they are elbow corners and checks whether enough angle difference exists for the upper and lower segments. If there are more than one candidate points for the elbow, Algorithm 4 selects the elbow point that gives the largest angle difference between the upper arm and the lower arm. If an elbow is found, the arm curve is divided into two pieces from the elbow and the corresponding angle is computed for each segment like in lower leg segments. If no elbow is detected then the angle is computed from the whole arm curve and the lower and upper arm are assigned the same angle value that is computed from the whole arm curve.

If the legs cross each other, as in Fig. 3(c), we cannot determine which one is left and which one is right. We use Algorithm 5 to detect whether the lower legs cross each other. If Algorithm 5 decides that the legs cross each other, then we swap left and right lower legs in the 2D pose extraction process.

Algorithm 5 goes over the lower parts of the silhouette, which are lower and upper legs by following the scan lines and produces a line segment list for the parts inside the silhouette. Then it analyzes the list of lines to decide whether lower legs cross each other. If we observe that the line lists at the beginning of the lower silhouette parts start with one line and becomes two lines for the remaining scan lines then we conclude that the legs cross each other. In such a case, we swap the angle values of the left lower leg with that of the right lower leg.

3.3.3. *Finding the foreshortening ratio of a body part*

We can find how much a segment is foreshortened based on the orthographic projection. However, we cannot detect the direction of the foreshortening. We get foreshortening direction from the user.

Algorithm 6 finds how much one or more body segments are foreshortened. The algorithm does not care about the direction of the foreshortening and finds the relative ratio, which is a measure

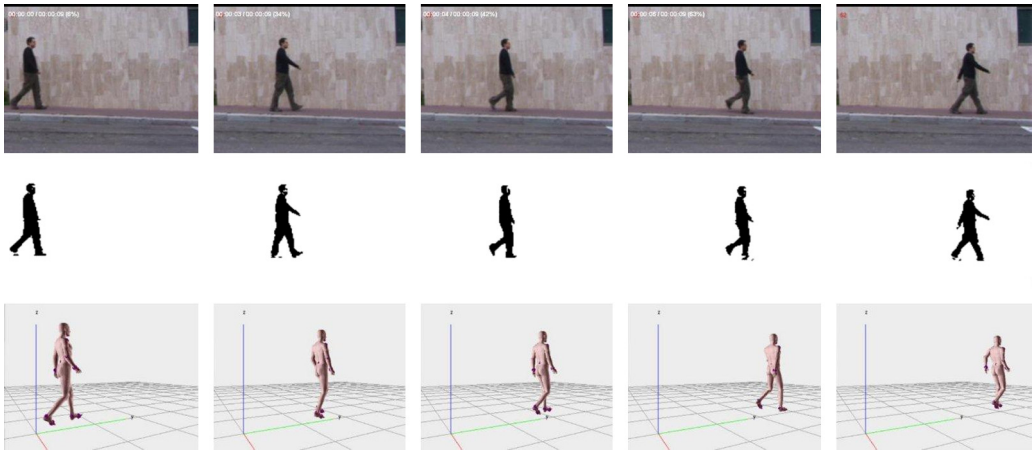

Fig. 4. The output of the implementation for the walking sequence. First row: sample images from the video sequence; second row: the performer's silhouette extracted; third row: the 3D human model pose obtained with the parameters extracted from the silhouette.

of the foreshortening angle. Algorithm 6 takes the fragment of human silhouette that contains the body segment to be searched as input. It tries to find the best ratio that covers the segment being searched and starts with an initial target ratio in the interval $[0, 1]$. Then, it iteratively narrows the target interval and finds the best ratio that covers the segment. At each iteration, the ratio interval is divided into a specified number of intervals and we measure how well the silhouette is covered by the segment for each ratio value that corresponds to each division. We find the division (ratio) that covers the silhouette best. We recursively continue to narrow the search interval and when the search interval is smaller than a threshold ratio, the algorithm returns the ratio that maximizes the fitting. Examples of the application of this algorithm are shown in Figs. 3(d) and 3(e).

3.4. 3D pose estimation

This stage estimates the 3D pose of the performer on the video. Given 2D joint coordinates and foreshortening direction of each body segment, a 3D pose can be constructed using orthographic projection. For the whole body, we take the orientation of the body as input from the user. The body can be in one of the six orientations: left, right, backward left, backward right, forward left, forward right. We also use horizontal foreshortening of the torso with the orientation of the body taken from the user to find how much the performer is rotated around the y-axis of the image. From the orientation, we also determine the left leg (arm) and the right leg (arm). Finally, we use the 3D pose found for each key frame to animate the character. We use linear interpolation to calculate the intermediate poses of the human body.

3.4.1. Finding 3D point coordinates

In this section, we explain how to find a corresponding point in an image according to orthographic projection as described in [26]. Let (X, Y, Z) be a point in the 3D world. The scaled orthographic projection coordinates (u, v) of the point are given by

$$\begin{pmatrix} u \\ v \end{pmatrix} = s \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \tag{4}$$

Let (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) be the two end points of a line with length l and the scaled orthographic projection coordinates of these two points are (u_1, v_1) and (u_2, v_2), respectively. If the scale factor s of the projection model is known, we can calculate the relative depth of the line denoted by ∂Z as shown in Eq. (5):

$$\begin{aligned} &l^2 = (X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2, \\ &(u_1 - u_2) = s(X_1 - X_2), \\ &(v_1 - v_2) = s(Y_1 - Y_2), \\ &\partial Z = (Z_1 - Z_2), \quad \text{and} \\ &\partial Z = \pm\sqrt{l^2 - \frac{((u_1 - u_2)^2 + (v_1 - v_2)^2)}{s^2}}. \end{aligned} \tag{5}$$

This analysis shows us how to compute the 3D correspondence of a point in the image as a function of the scale parameter s. In this work, we use the height of the human observed on the image to compute the scale parameter s. We use the parameter s found from the height and apply this parameter to all body parts to find 3D joint coordinates. For the computed value of s, two distinct solutions are possible since we do not know whether a segment on the image is in the front or at the rear [31].

4. Experimental results

We tried our implementation on two videos obtained from public resources. The first one shows a human walking [32] and the second one shows dancing [33]. The walking sequence consists of 82 frames and the dancing sequence consists of 200 frames. The output of our algorithm on these two sequences is illustrated in Figs. 4 and 5. (The videos for walking and dancing motions generated by our implementation can be found in http://www.cs.bilkent.edu.tr/~gudukbay/motion_capture.html.) The video frame which our method is applied to and the extracted silhouette from the video frame and the pose we constructed are depicted in the figure. We created an animation for each video by using key-framing technique. For the dancing video, we applied our method to 33 key-frames. For the walking video, we applied our method to 17 key-frames. While selecting the key-frames, we preferred the ones with low occlusion in order to get better results. We used linear interpolation to create intermediate frames. In both the dancing and walking videos we needed user interaction in a few frames to find the 2D joint coordinates because of high occlusion. We implemented the proposed system on a PC platform with Intel Celeron 2.70 GHz processor, 512 MB memory and Intel(R) 82852/82855 GM/GME graphics card, using Microsoft's C# .NET platform. It took nearly one second to find the 2D joint coordinates of a human body from a single video frame. The video frame size does not affect the performance significantly since we normalize the silhouette to a fixed size.

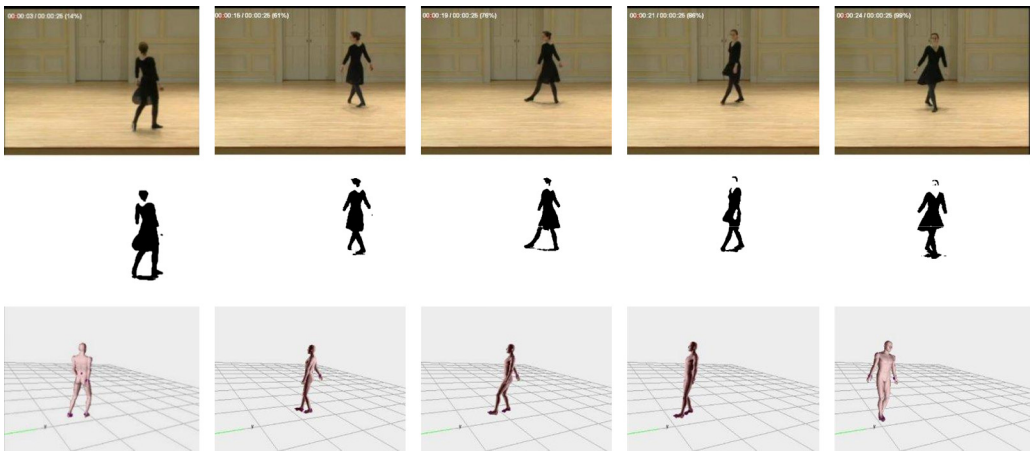

Fig. 5. The output of the implementation for the dancing sequence. First row: sample images from the video sequence; second row: the performer's silhouette extracted; third row: the 3D human model pose obtained with the parameters extracted from the silhouette.

In our implementation, we find the 2D joint coordinates from the images automatically and only require the user to enter the foreshortening direction and the body orientation. Our implementation only needs the user input for parameters that rarely change during a motion, so it requires minimal user intervention. In the walking video, the orientation of the body does not change during the whole video (82 frames). In the dancing video, the orientation of the body changes 7 times along the 200 frames. For the interactions required to specify the foreshortening direction, 52 changes occurred in the 82 frames of the walking video and 44 changes occurred in the 200 frames of the dancing video. In the dancing video, 8 user inputs are needed and in the walking video 7 user inputs are needed for the highly occluded parts in a few frames.

14 user inputs (joint coordinates) are needed for each video keyframe using Taylor's approach [26], which are determined automatically in our implementation.

If we use Taylor's approach in the walking video, we need $17 \times 14 + 52 = 290$ user inputs, where 17 is the number of keyframes in the animation, 14 is the number of joint coordinates specified for each frame, and 52 is the number of user interactions needed for specifying the foreshortening directions. In contrast, with our proposed work, $52 + 1 + 7 = 60$ user inputs are needed. There are 52 user interactions needed for the foreshortening directions, one user interaction needed for the orientation of the body and 7 user interactions needed for highly occluded parts.

If we use Taylor's approach in the dancing video, we need $33 \times 14 + 44 = 506$ user inputs, where 33 is the number of keyframes in the animation, 14 is the number of user interactions necessary for specifying the joint coordinates, and 44 is the number of user interactions needed for specifying the foreshortening directions. With our proposed framework, $44 + 7 + 8 = 59$ user inputs are adequate. Forty-four user interactions are required for specifying the foreshortening directions, 7 user interactions are required for specifying the orientation of the body, and 7 user interactions are required for specifying the joint angles of the highly occluded segments.

5. Conclusion

This paper proposes a framework for constructing the 3D human pose from a video sequence obtained from a single view. The proposed framework does not require camera calibration and requires minimal user intervention. Videos taken from public resources can be processed by the proposed framework, which assumes the input video has a static background, it has no significant perspective effects, and the performer is in an upright position. The proposed framework uses orthographic projection. By considering the foreshortening of the segments, the 3D pose of the human in the video can be reconstructed under orthographic projection. In contrast, the method proposed in [26] requires intensive user interaction; the user has to specify the joint coordinates on the images and the foreshortening direction for each segment. We calculate the joint coordinates automatically from the video images and succeed in reducing the user interaction for pose construction to 12–21% as compared to Taylor's approach [26].

By using the 3D poses constructed, we produced animations of human's walking and dancing. An animator has to control both the appearance and the movement of the characters. Since there are many degrees of freedom to be controlled, controlling the movement of characters is a difficult task that requires skill and labor. For this reason, the animators usually begin their work by sketching the coarse version of the movements on key poses. They rework and refine the key poses to produce the final animation [34]. Our proposed framework can help the professional animators produce an initial version of the motion, which then can be refined further.

One of the apparent results of our human pose estimation framework is to enable non-skilled computer users to use computer animation. The main advantage of our approach over the other approaches is the ability to create a 3D pose that can be easily mapped onto different characters, or modified to fit the needs of a specific animation. Since our approach requires less user interaction and has fewer constraints, it can be used to construct motion libraries from public resources. Temporal coherence can be used to increase the performance of the framework. Because when we are fitting the 2D model to the silhouette we search the body parts for angle ranges. With temporal coherence searching range can be reduced. This cannot be used for a video sequence without intervention. If occlusion occurs for a body part then the estimated angle will not be accurate so temporal coherence can not help after misestimated frames.

Acknowledgment

We are grateful to Miss Kirsten Ward for proofreading and suggestions.

Appendix A. Algorithms in pseudocode

This appendix provides Algorithms 1–6 used at different stages of our framework in order to make the work reproducible.

```
S ← PartialSilhouetteContainingSegmentsToBeSearched;
startAngle ← startAngle;
endAngle ← endAngle;
numberOfDivision ← 8;
minimumStepAngle ← 1°;
searchAngleInterval ← (endAngle - startAngle);
stepAngle ← (searchAngleInterval / numberOfDivision);
angleThatMaximizes ← (endAngle + startAngle)/2;
maximumSimilarity ← 0;
while searchAngleInterval > minimumStepAngle do
    angForBegin ← angleThatMaximizes − searchAngleInterval/2;
    if angForBegin < startAngle then
        angForBegin ← startAngle;
    end
    angForEnd ← angleThatMaximizes + searchAngleInterval/2;
    if angForEnd > endAngle then
        angForEnd ← endAngle;
    end
    if stepAngle is 0 then
        stepAngle ← minimumStepAngle;
    end
    angleForBodyPart ← angForBegin;
    while angleForBodyPart ⩽ angForEnd do
        currentModelPose ← DrawBodyPart();
        similarityResult ← MeasureSimilarity(S,
        currentModelPose);
        if similarityResult > maximumSimilarity then
            maximumSimilarity ← similarityResult;
            angleThatMaximizes ← angleForBodyPart;
        end
        angleForBodyPart ← angleForBodyPart + StepAngle;
    end
    searchAngleInterval ← 2 × (searchAngleInterval /
    numberOfDivision);
    stepAngle ← 2 × (stepAngle / numberOfDivision);
end
return angleThatMaximizes
```

Algorithm 1: The algorithm to determine how much the given segment(s) is rotated around the normal axis of the image.

```
S ← silhouette;
leftArmPoints ← GetLeftArmCurve(S);
rightArmPoints ← GetRightArmCurve(S);
leftLowerKneePoints ← GetLeftLowerKneeCurve(S);
rightLowerKneePoints ← GetRightLowerKneeCurve(S);
/* lla stands for LeftLowerArm                    */
/* lua stands for LeftUpperArm                    */
/* rla stands for RightLowerArm                   */
/* rua stands for RightUpperArm                   */
/* llk stands for LeftLowerKnee                   */
/* rlk stands for RightLowerKnee                  */
leftArmAngles ← FindArmAngles(leftArmPoints);
llaAngle ← leftArmAngles.Lower;
luaAngle ← leftArmAngles.Upper;
rightArmAngles ← FindArmAngles(rightArmPoints);
rlaAngle ← rightArmAngles.Lower;
ruaAngle ← rightArmAngles.Upper;
llkAngle ← FindAngle(leftLowerKneePoints);
rlkAngle ← FindAngle(rightLowerKneePoints);
llkLineList ← KneeScanLinesLeft(S);
rlkLineList ← KneeScanLinesRight(S);
if KneeCrossed(leftLowerKneeLineList, rightLowerKneeLineList) then
    Swap(llkAngle, rlkAngle);
end
return llaAngle, luaAngle, rlaAngle, ruaAngle, llkAngle, rlkAngle
```

Algorithm 2: The algorithm to find the limb angles.

```
lowerArmAngle ← 0;
upperArmAngle ← 0;
elbowCornerIndex ← FindElbowCorner(armPointList);
if elbowCornerIndex is −1 then
    angle ← FindAngle(armPointList);
    lowerArmAngle ← angle; upperArmAngle ← angle;
else
    startIndexUpper ← 0;
    endIndexUpper ← elbowCornerIndex;
    startIndexLower ← elbowCornerIndex;
    endIndexLower ← ArmPointList.Length −1;
    UpperArmPoints ← GetPoints(armPointList, startIndexUpper,
                               endIndexUpper);
    LowerArmPoints ← GetPoints(armPointList, startIndexLower,
                               endIndexLower);
    lowerArmAngle ← FindAngle(LowerArmPoints);
    upperArmAngle ← FindAngle(UpperArmPoints);
end
return lowerArmAngle, upperArmAngle
```

Algorithm 3: The algorithm to find the arm angle.

```
minimumCornerDistance ← 4;
minimumCornerAngleDifference ← 10;
listOfPossibleCorners ← CreateEmptyList();
for i ← minimumCornerDistance to armPointList.Length −1 do
    startIndexUpper ← 0;
    endIndexUpper ← i;
    startIndexLower ← i;
    endIndexLower ← armPointList.Length −1;
    upperArmPoints ← GetPoints(armPointList, startIndexUpper,
                               endIndexUpper);
    lowerArmPoints ← GetPoints(armPointList, startIndexLower,
                               endIndexLower);
    ArmAngles.Lower ← FindAngle(lowerArmPoints);
    ArmAngles.Upper ← FindAngle(upperArmPoints);
    cornerAngleDifference ← Absolute(ArmAngles.Upper −
                                     ArmAngles.Lower);
    if cornerAngleDifference ⩾ minimumCornerAngleDifference then
        Add(listOfPossibleCorners, i, cornerAngleDifference);
    end
end
cornerIndex ← −1;
if listOfPossibleCorners.Length ⩾ 0 then
    cornerIndex ←
    FindIndexOfMaxDifference(listOfPossibleCorners);
end
return cornerIndex
```

Algorithm 4: The algorithm to find the elbow corner.

```
/* lll stands for LeftLowerLeg                                    */
/* rll stands for RightLowerLeg                                   */
lllLineList ← LeftLowerLegLineList;
rllLineList ← RightLowerLegLineList;
OneKneeStarted ← false;
TwoKneeStarted ← false;
TwoKneeCrossed ← true;
for i ← 0 to lll.Length −1 do
    if lllLineList[i] intersects rllLineList[i] then
        if TwoKneeStarted then
            TwoKneeCrossed ← false;
            break;
        end
        OneKneeStarted ← true;
    else
        if OneKneeStarted then
            TwoKneeStarted ← true;
        else
            TwoKneeCrossed ← false;
            break;
        end
    end
end
if TwoKneeStarted is false then
    TwoKneeCrossed ← false;
end
return TwoKneeCrossed
```

Algorithm 5: The algorithm to detect whether the lower legs cross each other.

```
S ← partialSilhouetteContainingSegmentsToBeSearched;
beginPoint ← beginPoint;
endPoint ← beginPoint;
length ← DistanceOf(beginPoint, endPoint);
startRatio ← 0;
endRatio ← 1;
numberOfDivision ← 5;
minimumStepRatio ← 1/ length;
searchRatioInterval ← (endRatio − startRatio);
stepRatio ← (searchRatioInterval / numberOfDivision);
ratioThatMaximizes ← (endRatio + startRatio)/2;
maximumSimilarity ← 0;
while searchRatioInterval > minimumStepRatio do
    ratioForBegin ← ratioThatMaximizes − searchRatioInterval/2;
    if ratioForBegin < startRatio then
        ratioForBegin ← startRatio;
    end
    ratioForEnd ← ratioThatMaximizes + searchRatioInterval/2;
    if ratioForEnd > endRatio then
        ratioForEnd ← endRatio;
    end
    ratioForBodyPart ← ratioForBegin;
    while ratioForBodyPart ⩽ ratioForEnd do
        currentModelPose ← DrawBodyPart();
        similarityResult ← MeasureSimilarity(S, currentModelPose);
        if similarityResult > maximumSimilarity then
            maximumSimilarity ← similarityResult;
            ratioThatMaximizes ← ratioForBodyPart;
        end
        ratioForBodyPart ← ratioForBodyPart + stepRatio;
    end
    searchRatioInterval ← 2 × (searchRatioInterval / numberOfDivision);
    stepRatio ← 2 × (stepRatio / numberOfDivision);
end
return ratioThatMaximizes
```

Algorithm 6: The algorithm to find the absolute foreshortening.

Appendix B. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.dsp.2013.06.008.

References

[1] M. Gleicher, N. Ferrier, Evaluating video-based motion capture, in: Proceedings of Computer Animation, 2002, pp. 75–80.
[2] S. Kiss, Computer animation for articulated 3D characters, Technical Report, 2002-45, CTIT Technical Report Series, 2002.
[3] D. Thalmann, Physical, behavioral, and sensor-based animation, in: Proceedings of Graphicon, 1996, pp. 214–221.
[4] N.M. Thalmann, D. Thalmann, Computer animation, ACM Comput. Surv. 28 (1) (1996) 161–163.
[5] F. Perales, Human motion analysis and synthesis using computer vision and graphics techniques: State of art and applications, in: Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics, 2001.
[6] T.B. Moeslund, E. Granum, A survey of computer vision-based human motion capture, Comput. Vis. Image Underst. 81 (3) (2001) 231–268.
[7] S. Bryson, Virtual reality hardware, in: Implementing Virtual Reality, in: ACM SIGGRAPH Course Notes, vol. 43, 1993, pp. 1.3.16–1.3.24.
[8] H. Zhou, H. Hu, A survey-human movement tracking and stroke rehabilitation, Technical Report, No. 1744 - 8050, CSM-420, Department of Computer Sciences, University of Essex, UK, 1996.
[9] P.F. Felzenszwalb, D.P. Huttenlocher, Pictorial structures for object recognition, Int. J. Comput. Vis. 61 (1) (2005) 55–79.
[10] D. Ramanan, D.A. Forsyth, Automatic annotation of everyday movements, in: S. Thrun, L. Saul, B. Schölkopf (Eds.), Advances in Neural Information Processing Systems, vol. 16, MIT Press, Cambridge, MA, 2004.
[11] R. Rosales, M. Siddiqui, J. Alon, S. Sclaroff, Estimating 3D body pose using uncalibrated cameras, in: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR, vol. 1, 2001, pp. 821–827.
[12] G. Mori, J. Malik, Estimating human body configurations using shape context matching, in: Proceedings of European Conference on Computer Vision, ECCV, 2002, pp. 666–680.
[13] I.A. Kakadiaris, D.N. Metaxas, Model-based estimation of 3D human motion with occlusion based on active multi-viewpoint selection, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR, June 1996, pp. 81–87.
[14] I.A. Kakadiaris, D.N. Metaxas, Model-based estimation of 3D human motion, IEEE Trans. Pattern Anal. Mach. Intell. 22 (12) (2000) 1453–1459.
[15] G. Shakhnarovich, P.A. Viola, T. Darrell, Fast pose estimation with parameter-sensitive hashing, in: Proceedings of IEEE International Conference on Computer Vision, ICCV, 2003, pp. 750–759.
[16] L. Ren, G. Shakhnarovich, J.K. Hodgins, H. Pfister, P.A. Viola, Learning silhouette features for control of human motion, ACM Trans. Graph. 24 (4) (2005) 1303–1331.
[17] C. Bregler, J. Malik, Tracking people with twists and exponential maps, Technical Report, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR'98, June 1998, pp. 8–15.
[18] M. Ye, X. Wang, R. Yang, L. Ren, M. Pollefeys, Accurate 3D body pose estimation from a single depth image, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, November 2011, pp. 731–738.
[19] X. Wei, J. Chai, VideoMocap: Modeling physically realistic human motion from monocular video sequences, in: Proceedings of SIGGRAPH'2010, ACM Trans. Graph. 29 (4) (2010), article No. 42, 10 pp.
[20] M. Vondrak, L. Sigal, J. Hodgins, O. Jenkins, Video-based 3D motion capture through biped control, in: Proceedings of SIGGRAPH'2012, ACM Trans. Graph. 31 (4) (July 2012), article No. 2712.
[21] L. Ballan, G.M. Cortelazzo, Marker-less motion capture of skinned models in a four camera set-up using optical flow and silhouettes, in: Proceedings of 3DPVT'08 – the Fourth International Symposium on 3D Data Processing, Visualization and Transmission, Atlanta, GA, June 2008.
[22] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, H.-P. Seidel, Motion capture using joint skeleton tracking and surface estimation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR'09, 2009.
[23] B. Michoud, E. Guillou, S. Bouakaz, Real-time and markerless full-body human motion capture, in: Groupe de Travail Animation et Simulation, GTAS'07, Lyon, France, 2007.
[24] L. Gorelick, M. Blank, E. Shechtman, M. Irani, R. Basri, Actions as space–time shapes, IEEE Trans. Pattern Anal. Mach. Intell. 29 (12) (2007) 2247–2253.
[25] F. Ofli, E. Erzin, Y. Yemez, A.M. Tekalp, C.E. Erdem, A.T. Erdem, T. Abaci, M.K. Ozkan, Unsupervised dance figure analysis from video for dancing avatar animation, in: Proceedings of International Conference on Image Processing, ICIP, San Diego, CA, October 2008, pp. 12–15.
[26] C.J. Taylor, Reconstruction of articulated objects from point correspondences in a single uncalibrated image, Comput. Vis. Image Underst. 80 (3) (2000) 349–363.
[27] A. Memişoğlu, U. Güdükbay, B. Özgüç, Motion control for realistic walking behavior using inverse kinematics, in: Proceedings of the 3DTV-CON'07: Capture, Transmission, and Display of 3D Video, Kos, Greece, 2007.
[28] M.Ş. Yeşil, U. Güdükbay, Realistic rendering and animation of a multi-layered human body model, in: Proceedings of the Tenth International Conference on Information Visualization (IV), 2006, pp. 785–790.

[29] R. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, A system for video surveillance and monitoring: VSAM final report, Technical Report, CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May 2000.
[30] C.-L. Huang, C.-Y. Chung, A real-time model-based human motion tracking and analysis for human–computer interface systems, EURASIP J. Appl. Signal Process. 11 (2004) 1648–1662.
[31] V. Mamania, A. Shaji, S. Chandran, Markerless motion capture from monocular videos, in: Proceedings of Indian Conference on Computer Vision, Graphics and Image Processing, ICVGIP, 2004, pp. 126–132.
[32] Lena Gorelick, Moshe Blank, Eli Shechtman, The walking video, http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html, accessed at May 2013.
[33] Music Division, Library of Congress, The Dancing Video, An American Ballroom Companion, Video Directory, http://rs6.loc.gov/ammem/dihtml/divideos.html, accessed at May 2013.
[34] J. Davis, M. Agrawala, E. Chuang, Z. Popović, D. Salesin, A sketching interface for articulated figure animation, in: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Aire-la-Ville, 2003, pp. 320–328.

Uğur Güdükbay received a BSc degree in computer engineering from Middle East Technical University, Ankara, Turkey, in 1987. He received his MSc and PhD degrees, both in computer engineering and information science, from Bilkent University, Ankara, Turkey, in 1989 and 1994, respectively. Then, he conducted research as a postdoctoral fellow at the University of Pennsylvania, Human Modeling and Simulation Laboratory. Currently, he is an associate professor at Bilkent University, Department of Computer Engineering. His research interests include computer graphics (especially human modeling and animation, crowd simulation, visualization of complex graphical environments, virtual and augmented reality), computer vision, and computational geometry. He is a senior member of both IEEE and ACM.

İbrahim Demir received his BSc and MSc degrees in computer engineering from Bilkent University, Ankara, Turkey, in 2003 and 2006, respectively. Currently, he is a software engineer at Turkcell, İstanbul, Turkey. His research interests include computer graphics and computer vision, including processing motion capture data and motion reconstruction.

Yiğithan Dedeoğlu received his BSc and MSc degrees in computer engineering, Ankara, Turkey, in 2002 and 2004, respectively. During his MSc studies, he worked as a software development engineer at Synectics Ltd. Research & Development Group at Bilkent University. Currently, he is a software development engineer at Microsoft, USA. His research interests include computer vision, including motion detection, object tracking and classification, human action recognition, camera sabotage detection, and fire and smoke detection.