Integrating Human Body MoCaps into Blender using RGB Images

Jordi Sanchez-Riera and Francesc Moreno-Noguer

Institut de Robòtica i Informàtica Industrial, CSIC-UPC
08028, Barcelona, Spain
Email: {jsanchez,fmoreno}@iri.upc.edu

Abstract—**Reducing the complexity and cost of a Motion Capture (MoCap) system has been of great interest in recent years. Unlike other systems that use depth range cameras, we present an algorithm that is capable of working as a MoCap system with a single Red-Green-Blue (RGB) camera, and it is completely integrated in an off-the-shelf rendering software. This makes our system easily deployable in outdoor and unconstrained scenarios. Our approach builds upon three main modules. First, given solely one input RGB image, we estimate 2D body pose; the second module estimates the 3D human pose from the previously calculated 2D coordinates, and the last module calculates the necessary rotations of the joints given the goal 3D point coordinates and the 3D virtual human model. We quantitatively evaluate the first two modules using synthetic images, and provide qualitative results of the overall system with real images recorded from a webcam.**

Keywords–MoCap; 2D, 3D human pose estimation; Synthetic human model; Action mimic.

I. Introduction

Motion Capture (MoCap) systems are used in industry and research to record real motions. The applications of such systems span from animating virtual characters or facial expressions, navigating into Virtual Reality (VR) environments, to modeling human-human/robot/object interactions. Professional MoCap systems are expensive, complex to use and need some dedicated space to record the motions, usually with multiple cameras. More modern systems just require to wear a suit which has reflective markers or motion sensors [1] [2]. These systems store the motions into files with a standard format that can be shared with other applications. However, due to the complexity of recording and processing the data from such systems, and that not everyone can afford to acquire a MoCap suite, it is not easy to find motion files processed by third parties. Or, even in the case we can find public repositories with motion files, as in [3] or [4], the motions we can find may not be those we need.

In order to make the MoCap recordings more affordable, there have been several attempts to find alternatives to reduce the complexity and time to process the recorded data. A good example can be found in [5], where the authors only need two calibrated cameras to infer 3D points given by a set of reflective markers on the human body. To eliminate the burden of having to wear body/cloth markers, Ganapathi et al. [6] propose a new system that uses depth images, therefore 3D locations come directly from the camera sensor device. This method, however, still needs a camera calibration process, which is a tedious task. More recently, Mehta et al. [7] eliminated the need of camera calibration, proposing a system able to infer 3D joint locations from a single RGB camera. At the same time, the irruption of Kinect camera has encouraged homebrew developers to program algorithms using the Application Program Interface (API) device library to achieve an inexpensive MoCap system for body [8] or faces [9]. Unfortunately, the official API library is only available on Windows platforms and some of the free alternative libraries are obsolete.

Similar to Mehta et al. [7], we propose an algorithm that is able to infer 3D body joint locations from a single RGB camera, and at the same time, we integrate it to a Blender 3D modeling software [10] that allows us to generate realistic renders using Makehuman [11] 3D human model. This allows easy saving and exporting captured motions into an industry standard motion file, which could be used by other software applications. Moreover, eliminating the need of a depth sensor device, as opposed to [6] [8], we can use our system both indoors and outdoors. As illustrated in Figure 1, our algorithm is composed of three modules. The first module takes the images from the camera and estimates the 2D joint articulations of the body. We then estimate the 3D human pose locations from the 2D detected joints, and finally the third module calculates the rotation of each joint to map from the virtual 3D human body joints to the 3D pose locations estimated by the second module.

The rest of the paper is organized as follows. In the next section, we discuss the related work for the first and second implemented modules. In Section III, we describe the developed algorithm as well as its integration to Blender. Then, in Section IV, we present the synthetic and real performed experiments, and, finally, in Section V, we draw the conclusions.

II. Related Work

One of the key parts for a MoCap system is to have a reliable human pose detector. There are many works in literature on human pose estimators, that is why in this section, we will only review the most significant in 2D human pose detection and 3D human pose estimation.

A. 2D Human Pose Detection

When estimating 2D human pose from a single RGB image, there are two possible different approaches. One, known as bottom-up approach, consists into find the body joint locations and then trying to reason about the best configuration links that matches the observed body. The other one, known as top-down approach, starts from localizing the whole body region and later detecting the several body joints. A very popular bottom-up algorithm was introduced by Wei et al. [12],

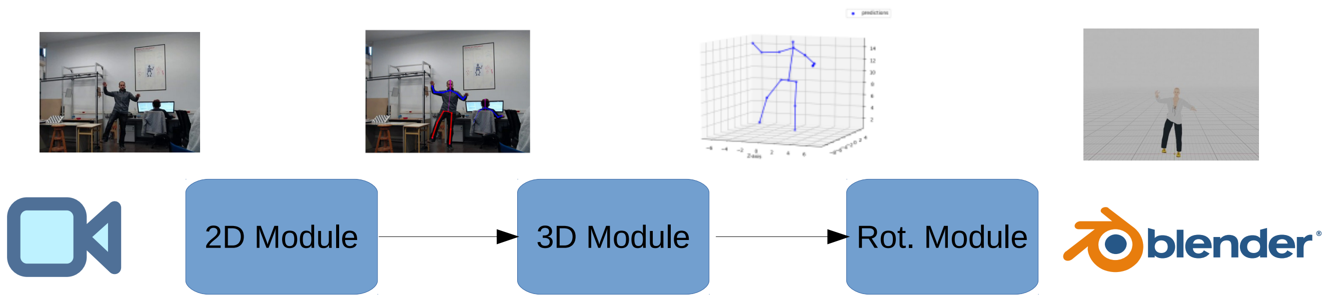


Figure 1. Pipeline of the proposed MoCap system.

and was improved by adding body parts optical flow in Cao et al. [13]. Both algorithms can run on several platforms, e.g., PC, phone, tablet, at very high frame rates. Top-bottom approaches can also detect 2D poses of multiple persons at high frame rates. Most of them first run a human detector [14] to localize body regions, however, these algorithms are prone to have problems when two persons overlap with each other. The most popular top-bottom algorithms are [15] [16], and their respective improved versions [17] [18]. Finally, another very popular approach [19] combines multiple bottom-up and top-down layers together, named hourglass, to detect the human pose at multiple resolutions. For our system, we decide to use the AlphaPose [16] method because it outperforms the other algorithms presented in this section, can also run in real time, and the skeleton that is retrieved is more similar to our 3D human skeleton model used in Blender, see Figure 2 a, b.

B. 3D Human Pose Estimation

Estimating 3D human pose from a single RGB image is an important challenge. Most approaches assume that 2D joint locations are already known [20]–[22]. Some other approaches make use of additional extra cues, such as descriptors [23], stereo information [24] or even part models [25]. However, the most common used approach consists of combining 2D and 3D detections to make the 3D pose estimation more robust to errors [26]. One of the problems that arises when training these networks is the lack of labeled data. It is not easy to find ground truth for 3D human poses. For this reason, Zhou et al. [27] propose a method to combine labeled images with images in the wild. Chen et al. [28] go one step further and estimate directly 3D human pose from a single image using a synthetic dataset of 5M labeled images. These kind of networks are quite complex [29], and using extra information such as temporal correlations, can help to improve their performance [30]. Most recent algorithms, not only can find a 3D human pose estimate, but also can recover the whole body mesh [31] or even the shape parameters [32]. We decide to use the method described in Martinez et al. [20] due to its simplicity for training. The network described is relatively simple, and can be trained with thousands of samples instead of millions. Moreover, the network inference is very fast.

III. Method

The proposed MoCap system is divided into several interconnected modules. These modules will process the RGB images coming from a webcam, and finally will control a 3D human model through the Blender software. We first explain the modules responsible to infer 3D body joint positions from a single image. Then, we describe how these 3D joints are transformed into rotations for our 3D human model and, finally, how these rotations are passed to the render software.

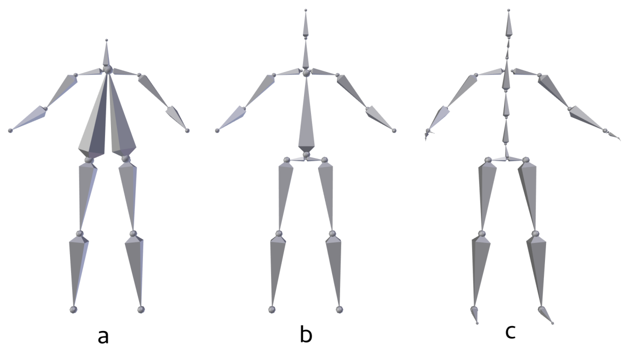


Figure 2. Different skeletons configurations. a) skeleton given by CPM [12] method, b) Alpha Pose skeleton used to compute angle rotations, c) skeleton used for Makehuman model to which captured motion is transferred.

A. Find 2D Joint Locations

We follow Alpha Pose [16] to obtain 2D human pose estimations from a single RGB image. The method is fast, outperforms other state-of-the-art algorithms, and the returned body pose is similar to the Makehuman 3D human model skeleton that we use in Blender. The method starts from some human region proposals, then these region proposals are passed through three different modules. The first one is the symmetric spatial transfer network that generates a set of pose proposals. The second one, a parametric pose non-maximum-suppression module, selects the most plausible pose estimations. And finally, the third one, the pose-guided proposals generator, is used to augment the training data and improve the network performance.

B. Infer 3D Joint Coordinates

Given a set of 2D points $x \in \mathbb{R}^2$ obtained in the previous module, we want to find a regression function that estimates the 3D points $y \in \mathbb{R}^3$ and minimizes the error over a set of 3D body poses. The regression function will be modeled by a neural network defined in Martinez et al. [20]. This network is composed of two consecutive blocks that contain a linear layer with batch normalization and a Rectified Linear Unit (ReLU) followed by a dropout layer. We use default parameters to train this network.

C. Animate 3D Human Model

Our 3D human model is controlled by a hierarchical structure called skeleton. This skeleton can be seen as a directed graph, where there is a root node and none or several children for each node. Each node is a segment where its start position defines the rotation pivot point and the end position is the start of the child node. For each node, also named bone or joint, a bind matrix M_{bj} encodes the joint position and rotation of the skeleton at rest pose. A pose matrix M_{pj} encodes the amount of rotation of each joint with respect to the rest of the skeleton pose. Thus, a skeleton pose $P(\theta)$ will be defined by a set of rotation parameters $\theta \in \mathbb{R}^{3K}$ for each one of the K joints that have direct correspondence with the 3D virtual model, Figure 2 c.

We want to calculate, for each joint of our skeleton, the rotation that matches a set of estimated 3D joint locations. The 3D joint locations, Figure 2 b, and our model skeleton, Figure 2 c, can have a different structure. Therefore, we can only find rotations for the skeleton joints that are equivalent in both structures. In order to calculate the rotations of each joint, we will define a source vector v_s and a target vector v_t. The source vector v_s will go from the the parent of the joint to the beginning of the child of the joint. The target vector v_t will be defined by the 3D point coordinates. If $v = v_s \times v_t$, $s = ||v||$ and $c = v_s \cdot v_t$, the rotation matrix to match the two vectors v_s, v_t is defined by:

$$R = I + [v]_x + [v]_x^2 \frac{1-c}{s^2} , \qquad (1)$$

where $[v]_x$ is the skew-symmetric cross product matrix of v.

Before we can calculate the rotation matrix, it is necessary to have the two sets of 3D points in the same coordinate system. Therefore, the skeleton 3D joint locations need to be transformed from local X_j^L coordinates to world X_j^W coordinates. In the case that the skeleton joint has no father, we will use (2), otherwise we will use (3), where M_{Tf} is the M_T matrix already calculated for the father of the current bone.

$$X_j^W = M_{bj} \cdot M_{pj} \cdot X_j^L = M_T \cdot X_j^L \qquad (2)$$

$$X_j^W = M_{Tf} \cdot M_{bf}^{-1} \cdot M_{bj} \cdot M_{pj} \cdot X_j^L \qquad (3)$$

Finally, to obtain the skeleton joint coordinates in Blender coordinates, we need to use the defined world matrix M_w.

$$X_j^G = M_w \cdot X_j^W \qquad (4)$$

D. Transfer Joint Rotations to Blender 3D Human Model

The final goal of the algorithm is to be able to transfer detected motion human poses from a videos or a webcam to a 3D human model in Blender. This will allow to store the motion into MoCap files or just simply use it as it is. To this end, we design a communication protocol between Blender and the image stream, using ZMQ library [33]. On one hand, we will have a process that will take the images from the stream and calculate the 3D joint coordinates of the body. On the other hand, we will have a Python script that will take the 3D joint locations from the stream as well as the rest of the 3D joint locations of the 3D human model and will calculate the body joint rotations. Therefore, we will use a server-client structure where the server will provide 3D joint locations at desired frame rate, while the client will be limited to listen to the data from the server.

IV. Experiments

We perform two different kinds of experiments. We generate two sets of synthetic data to train and evaluate the 3D human pose estimation, and then, we record several real sequences with a webcam to evaluate the whole algorithm performance.

A. Generate synthetic data

For training and evaluation purposes, we generate two different kind of datasets with Blender [10] and Makehuman [11]. The first dataset is used to train the 3D pose estimation module, while the second dataset is used to evaluate the overall performance of the proposed MoCap system. Note that for the 2D joint estimation module, a dataset is not needed since we use the weights trained from [16].

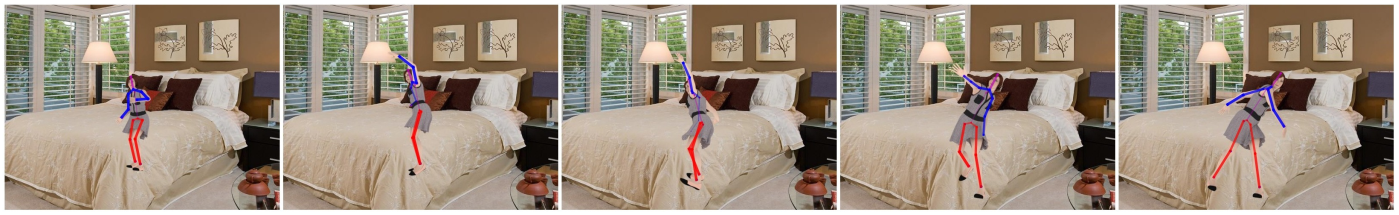
The first dataset consists of 6 human models (3 men and 3 women) with different shapes and sizes, performing a total of 54 motions obtained from Mixamo website [3]. Therefore, we have a total of 324 sequences of approximately 100 frames each. For each sequence, the ground truth for 3D joint locations as well as their 2D projections on a camera image are stored. We set the camera resolution to be of 640×480 for both datasets.

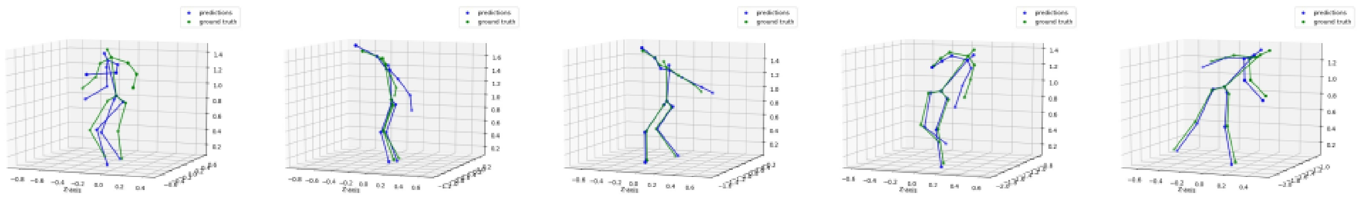
For the second dataset, we use 4 human models (2 men and 2 women) different from the ones used in the first dataset. We also use 10 motion sequences that were not included in the first dataset. In this case, the information gathered includes also the render RGB images of the sequences, apart from the 3D and 2D ground truth locations. To make the images more close to the real world, in each generated sequence, a random image is added as a background.

We also record five real sequences with a webcam at 640×480 image resolution to evaluate qualitatively the proposed MoCap system. Each one of the sequences is about 30 seconds long with a person performing different movements in the center of the image. The recorded images also have a person in the background of the scene, therefore, this person detection is removed from the image with a simple post-processing algorithm consisting in keeping the person with the biggest bounding box in the center of the image.

B. Evaluation

The images generated in the second dataset are passed to the Alpha Pose [16] detector network. This network returns the estimated 2D locations of 16 body joints, see Figure 2. The estimated locations are compared with ground truth locations using the PCKh@0.5 [34] measure. This measure calculates the percentage of correct keypoints when the threshold is 50% of the head bone link. The curves for the values of each action can be seen in Figure 4. We can observe that most of the actions have similar performance except for two actions: "teeter" and "picking up". In the case of "teeter" action, the motions of arms and legs are small but fast, making it seem like the person is shaking and this fact provokes several


(a) 2D joint detections



(b) 3D pose estimation

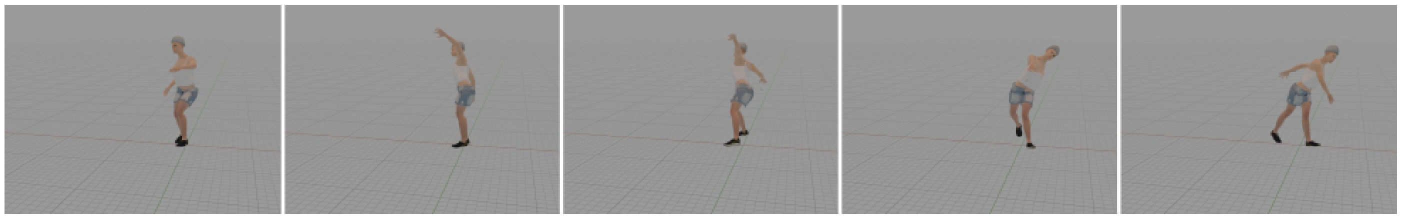

(c) Render 3D human model

Figure 3. Proposed algorithm on a synthetic sequence. a) 2D joint detections, b) ground truth 3D joint positions (in green) and the correspondent estimations by the 3D joint detection module (in blue), c) render images of the 3D human model. The input images are synthetized with Blender software. Ground truth is available.

TABLE I. MEAN PER JOINT POSITION ERROR (MPJPE) (m) FOR EACH SEQUENCE MOTION IN THE SECOND DATASET

	boxing	goalie throw	jumping jacks	looking around	picking up	talking	teeter	walking	walking 2	zombie kicking	average
error(m)	0.0961	0.0989	0.1299	0.1585	0.1399	0.0952	0.1416	0.1621	0.1535	0.1288	0.1304

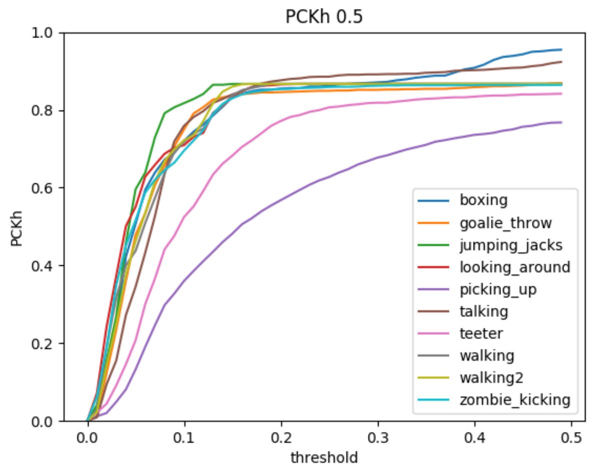
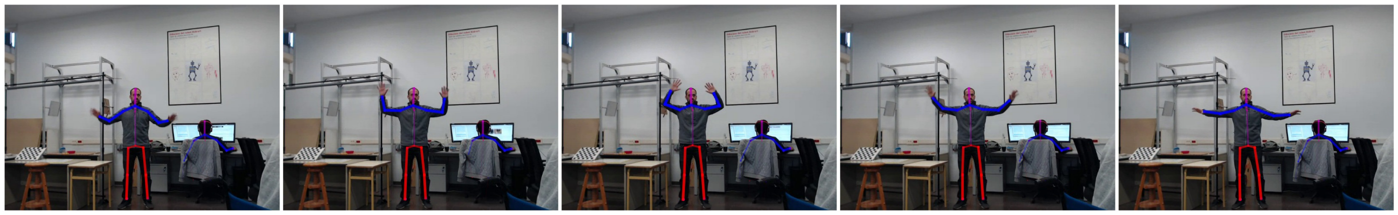


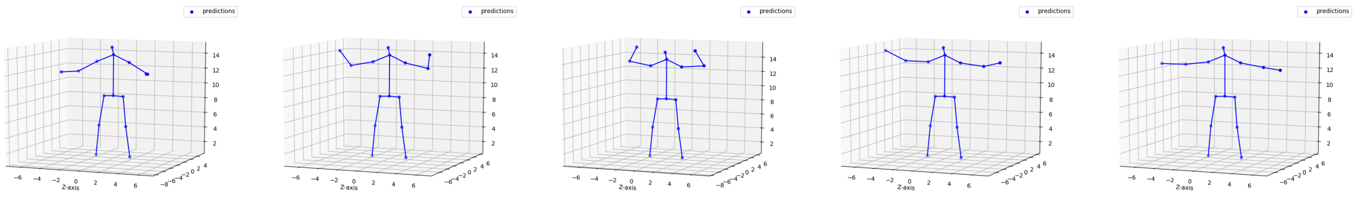
Figure 4. Percentage of correct keypoints at 50% of the head bone link for the different actions generated in the second dataset.

misdetections. In the case of ”picking up” action, the model bends the upper part of the body occluding the other bottom half, making the 2D estimation module fail. However, the overall performance among all the actions is quite accurate.

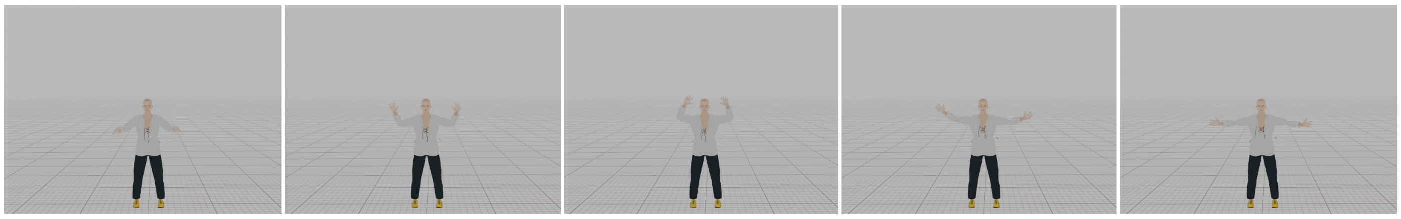
Unlike the 2D joint detection module, the 3D pose estimation module is trained from the scratch. The reason for that is because we want to express the 3D joint positions with the Blender coordinate system for a simpler calculation of rotations. Therefore, the ground truth 2D and 3D joint locations are normalized according the new generated training dataset. This means that all sequences are reoriented taking the ”hips” joints as reference, and values for all joints are rescaled to values from 0 to 1. Once the 3D joint estimation network is trained and weights are obtained, we proceed to evaluate the same 10 sequences as before. In this case, to evaluate the performance of the network, we use the Mean Per Joint Position Error (MPJPE) given in meters [35]. For evaluation, the ground truth 2D joint locations are replaced by the estimations obtained in the 2D joint estimation module. The results can be observed in Table I. The first thing to notice is that, while in the 2D detection model the actions ”teeter” and ”picking up” are the ones with worst performance, in the current 3D detection module the worst performance actions



(a) 2D joint detections



(b) 3D pose estimation



(c) 3D human model render

Figure 5. Results of the presented algorithm on the first real sequence.

are ”walking” and ”walking 2”. Therefore, the errors from previous 2D detection module are not propagated as we could expect. The action with less error correspond to ”boxing”, which is the action where the model has less joints moving. The mean performance of all the actions is very low, with an error of 0.13 meters.

In Figure 3, we show the results for 5 frames of the sequence ”goalie throw” for a woman actor. In each row, we present the results for each one of the proposed modules. The top row shows the results of the Alpha Pose algorithm. In the middle row, we show the ground truth 3D coordinates (in green), and the 3D estimated joint values (in blue). Finally, in the bottom row, we show a render model in Blender. We can observe that the original motion is very close to the render motion. In Figures 5 and 6, we show the results for two real sequences, when the actor is performing some random movements in the center of the scene. We can observe in the first row, that in the recorded images our 2D human pose detection module is able to find two different persons in the scene. Since for our proposed MoCap, we want only to focus on one person, we apply a simple image processing consisting in keeping biggest bounding box near to the center of the image. In the second row, we show the inferred 3D human position for the only person that we want to detect. In the third row, we show our virtual 3D model once calculated rotations are applied.

V. Conclusion

We presented a system capable of performing as a MoCap system with only a single RGB camera, with free source software that can run on several platforms. The system is based on three components that calculate 2D human body joint locations, then, from these locations, infer the 3D joint world coordinates, and finally, the 3D joints are transformed to joint body rotations for our virtual human model. The first two components are evaluated quantitatively with synthetic data, while the third component is only evaluated qualitatively. The overall system is also evaluated in real video images, with several sequences performed by a person. We show that we can mimic the movements of a person, with the potential to run the whole system in real time. In the future, we could use this MoCap system to perform dedicated actions for any kind of topic, and extract several ground truth data like in [36].

Acknowledgment

This work is supported by the Spanish MiNeCo under projects HuMoUR TIN2017-90086-R and Maria de Maeztu Seal of Excellence MDM-2016-0656. We also thank Marta Altarriba Fatsini for her support derivating the formulas to compute skeleton angle rotations.

References

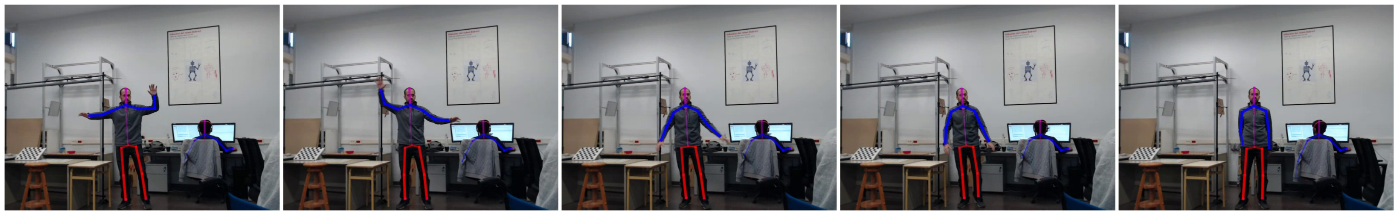
[1] “XSens: MotionCapture,” 2019, URL: https://www.xsens.com/ [accessed: February, 2020].

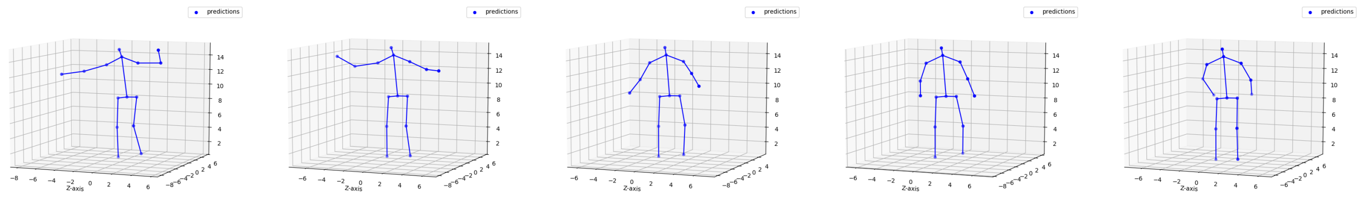
[2] “Vicon: MotionCapture,” 2019, URL: https://www.vicon.com/ [accessed: February, 2020].

[3] “Adobe Mixamo,” 2019, URL: https://www.mixamo.com/ [accessed: February, 2020].

[4] “CMU Motion Files,” 2019, URL: http://mocap.cs.cmu.edu/ [accessed: February, 2020].

[5] J. Chai and J. K. Hodgins, “Performance animation from low-dimensional control signals,” in SIGGRAPH, vol. 24, no. 3, 2005.


(a) 2D joint detections



(b) 3D pose estimation

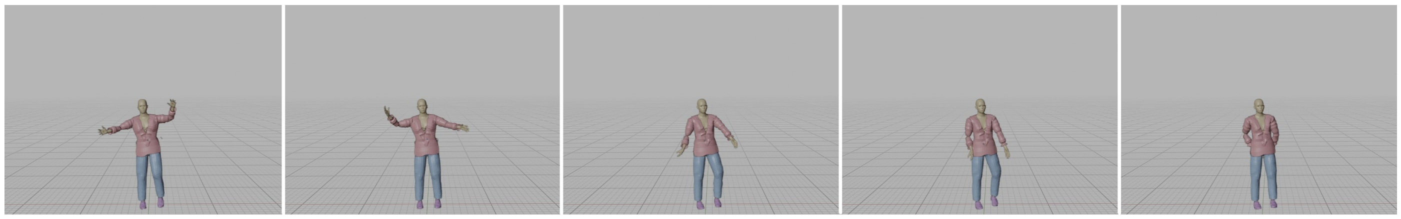

(c) 3D human model render

Figure 6. Results of the presented algorithm on the second real sequence.

[6] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, “Real-time human pose tracking from range data,” in ECCV, 2012.

[7] D. Mehta et al., “Vnect: Real-time 3d human pose estimation with a single rgb camera,” in SIGGRAPH, vol. 36, no. 4, 2017.

[8] “Homebrew Mocap Studio,” 2019, URL: https://www.youtube.com/watch?v=1UPZtS5LVvw [accessed: February, 2020].

[9] “Face Mocap Studio,” 2019, URL: https://brekel.com/brekel-pro-face-2/ [accessed: February, 2020].

[10] “Blender,” 2019, URL: https://www.blender.org/ [accessed: February, 2020].

[11] “Makehuman,” 2019, URL: http://www.makehumancommunity.org/ [accessed: February, 2020].

[12] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, “Convolutional pose machines,” in CVPR, 2016.

[13] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, “Realtime multi-person 2d pose estimation using part affinity fields,” in CVPR, 2017.

[14] R. Girshick, “Fast R-CNN,” in ICCV, 2015.

[15] L. Pishchulin et al., “Deepcut: Joint subset partition and labeling for multi person pose estimation,” in CVPR, 2016.

[16] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu, “RMPE: Regional multi-person pose estimation,” in ICCV, 2017.

[17] E. Insafutdinov et al., “Arttrack: Articulated multi-person tracking in the wild,” in CVPR, 2017.

[18] Y. Xiu, J. Li, H. Wang, Y. Fang, and C. Lu, “Pose Flow: Efficient online pose tracking,” in BMVC, 2018.

[19] A. Newell, K. Yang, and J. Deng, “Stacked hourglass networks for human pose estimation,” in ECCV, 2016.

[20] J. Martinez, R. Hossain, J. Romero, and J. J. Little, “A simple yet effective baseline for 3d human pose estimation,” in ICCV, 2017.

[21] F. Moreno-Noguer, “3d human pose estimation from a single image via distance matrix regression,” in CVPR, 2017.

[22] E. Simo-Serra, C. Torras, and F. Moreno-Noguer, “3d human pose tracking priors using geodesic mixture models,” in IJCV, vol. 122, no. 2, 2017, pp. 388–408.

[23] E. Simo-Serra, C. Torras, and F. Moreno-Noguer, “DaLI: Deformation and light invariant descriptor,” in IJCV, vol. 115, no. 2, 2015.

[24] J. Sanchez-Riera, J. Cech, and R. Horaud, “Robust spatiotemporal stereo for dynamic scenes,” in ICPR, 2012.

[25] E. Trulls, S. Tsogkas, I. Kokkinos, A. Sanfeliu, and F. Moreno-Noguer, “Segmentation-aware deformable part models,” in CVPR, 2014.

[26] C.-H. Chen and D. Ramanan, “3d human pose estimation = 2d pose estimation + matching,” in CVPR, 2017.

[27] X. Zhou, Q. Huang, X. Sun, X. Xue, and Y. Wei, “Towards 3d human pose estimation in the wild: A weakly-supervised approach,” in ICCV, 2017.

[28] W. Chen et al., “Synthesizing training images for boosting human 3d pose estimation,” in 3DV, 2016.

[29] X. Sun, B. Xiao, F. Wei, S. Liang, and Y. Wei, “Integral human pose regression,” in ECCV, 2018.

[30] M. Lin, L. Lin, X. Liang, K. Wang, and H. Chen, “Recurrent 3d pose sequence machines,” in CVPR, 2017.

[31] R. A. Güler, N. Neverova, and I. Kokkinos, “DensePose: Dense Human Pose Estimation In The Wild,” in CVPR, 2018.

[32] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik, “End-to-end recovery of human shape and pose,” in CVPR, 2018.

[33] “ZMQ,” 2019, URL: https://zeromq.org [accessed: February, 2020].

[34] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, “2d human pose estimation: New benchmark and state of the art analysis,” in CVPR, 2014.

[35] L. Sigal, A. Balan, and M. J. Black, “HumanEva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion,” in IJCV, vol. 87, no. 1, 2010.

[36] A. Pumarola, J. Sanchez-Riera, G. P. T. Choi, A. Sanfeliu, and F. Moreno-Noguer, “3dpeople: Modeling the geometry of dressed humans,” in ICCV, 2019.