FLEX: Extrinsic Parameters-free Multi-view 3D Human Motion Reconstruction Supplementary Materials

Brian Gordon*, Sigal Raab*, Guy Azov,
Raja Giryes, and Daniel Cohen-Or

Tel Aviv University
briangordon@mail.tau.ac.il, sigalraab@tauex.tau.ac.il,
guyazov@mail.tau.ac.il, {raja,dcor}@tauex.tau.ac.il

1 Outline

The following document is constructed in the following way. Section 2 describes how to access our model's code, and Section 3 provides additional media, namely figures that have not been included in the main paper due to page limitation, and a description of video files that are available at our project page[1]. In Section 4 we describe an improvement in the skeleton structure, and in Section 5 we detail the internals of our architecture. Section 6 thoroughly describes the datasets and various data aspects of our work, and finally, Section 7 presents technical details related to camera parameters.

2 Code

Our code, together with usage instructions, is available on our project page[1]. The reader is encouraged to run the code and witness the reproducibility of our model.

3 Additional visualizations

On our project page[1], the reader can find attached video files. The reader is encouraged to browse the video files in full-screen size. Here is their description:

- A clip describing our work: clip.mp4
- Video files showing our results on the Human3.6M dataset: Human36M*.mp4
- Video files showing our results on the KTH multi-view Football II dataset: KTH_football.mp4
- Video files comparing MotioNet (single-view) and Iskakov *et al*. [9] results versus ours: MotioNet_comparison.mp4 and Iskakov_comparison.mp4, respectively.

* equal contribution.
[1] Project page: https://briang13.github.io/FLEX

Note that we present only results that use input obtained by 2D estimation (as opposed to ground truth). Thus, our input is affected by occlusion and blur. Yet, we are able to mitigate the noisy input by exploiting multi-view data, in an ep-free fashion.

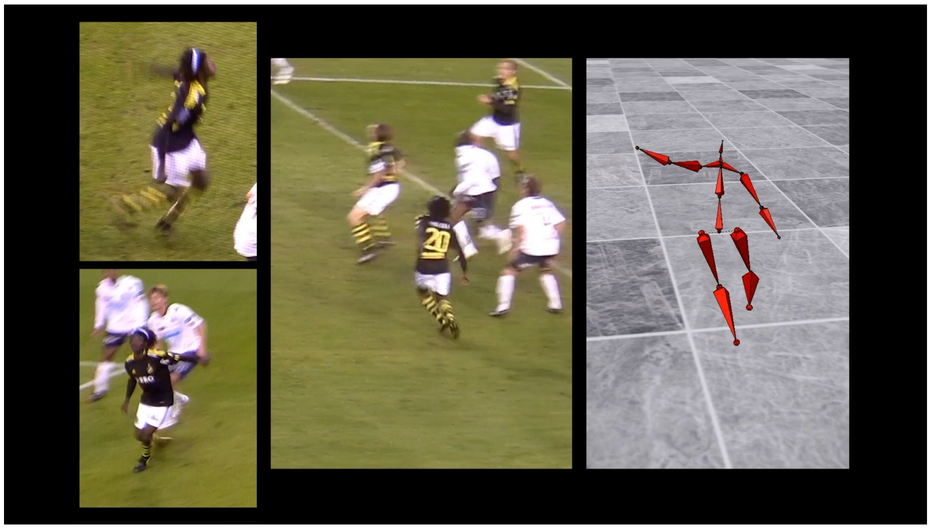

Fig. 1: Our algorithm is able to grasp fine details. The player's left hand cannot be seen in the center view and is blurred in the left views. Yet, our model accurately reconstructs it.

In Figure 1 we show how our algorithm is able to grasp fine details. The player's left hand cannot be seen in the center view and is blurred in the left views. Yet, our model accurately reconstructs it.

In Figures 8 and 9, we show additional results on the Human3.6M and KTH Football multi-view II datasets. Each row depicts three views of one time frame. To the right of each image we place a reconstructed rig. Figures 10 to 12 are enlarged versions of the figures shown in the main paper.

Figure 2 shows our recording setup for creation of synthetic data. Note the depicted cameras, that dynamically move in the scene.

In Figure 3 we depict qualitative results for a scene with two macarena dancers. We emphasize several viewpoints where the 2D backbone attains large errors. Yet, FLEX is able to compensate for these errors by fusing multi-view information. Figures 13 and 14 depict 2D joint locations estimated by the Alpha-Pose [7] backbone. A close look at these figures shows that many of the estimated locations are inaccurate, e.g., a hand of one subject is confused with the hand of the other subject. Even though the number of 2D errors is large, our algorithm is able to reconstruct the characters correctly.

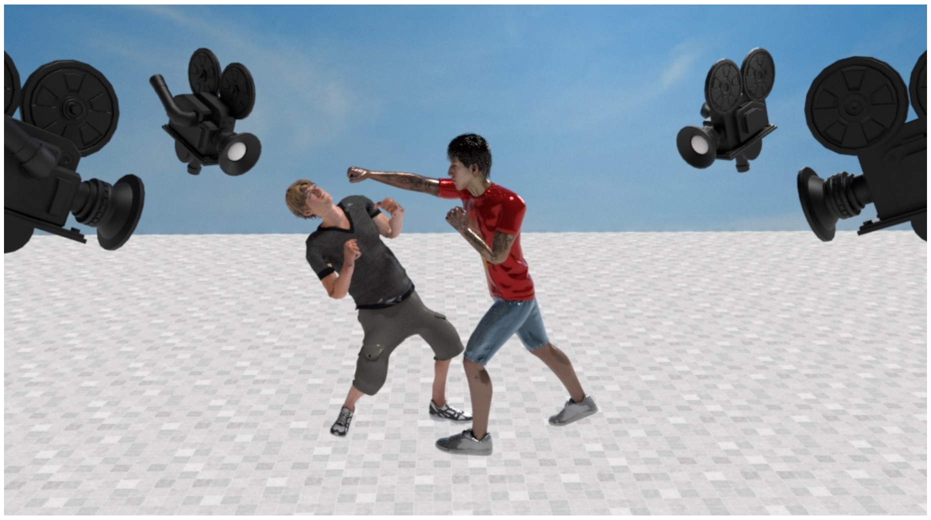

Fig. 2: Our "synthetic studio" created using Blender [5] software with Mixamo [1] 3D characters. Two interacting characters are captured by multiple dynamic cameras and rendered into multiple video streams.

4 Skeleton

To better reconstruct the motion in a given video stream, we modify the skeleton connectivity used in our baseline [15] (Figure 4). The root and neck joints of the baseline skeleton are both rigidly attached to the three bones neighboring each of them. This rigid connectivity constrains the skeleton, e.g., a motion where each shoulder moves forward and the neck moves to the right is impossible. In order to remove this constraint we add joints that overlap the root and the neck, hence enabling the neighboring bones to move independently of each other.

The new skeleton connectivity better matches the Human3.6M rotation angles ground-truth, thus, it better matches the way the dataset motions were captured. The new skeleton improves the mean per joint position error (MPJPE) both in the multi-view setting and the monocular case. The improvements are by ∼4mm and ∼6mm for monocular and four cameras setting, respectively.

5 Architecture details

The architectural blocks in our implementation are the multi-view feature fusion layers F_S and F_Q, the two encoders, E_S and E_Q, a forward kinematics layer FK and a discriminator D. Our discriminator D is a linear component that contains two convolution layers and one fully connected layer. We base it on Kanazawa *et al.* [10]). The FK layer is based on Villegas *et al.* [17].

There are two novel building blocks contained in the new fusion layers, F_S and F_Q. The first is a multi-view convolutional layer; that is, a convolution that is aware of features stemming from multiple views as well as multiple frames.

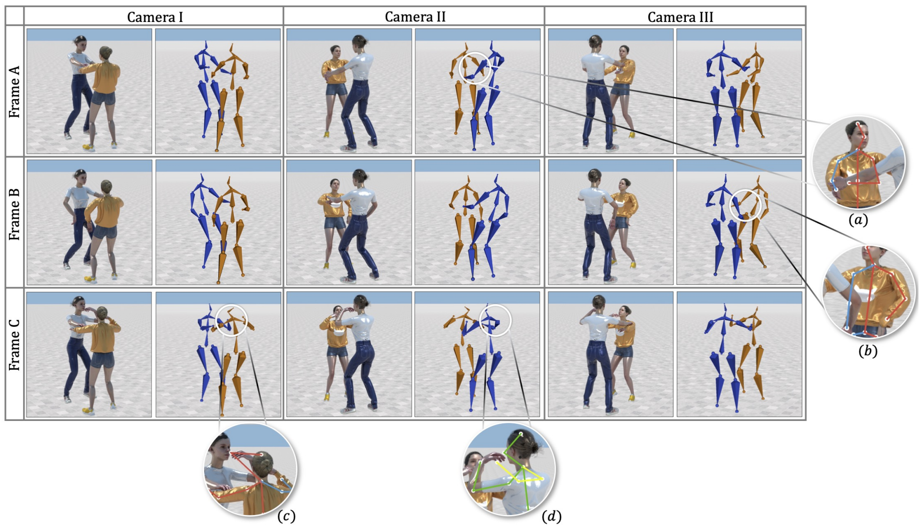

Fig. 3: Our results on multi-person synthetic videos, picturing two Macarena dancers. Some of the 2D joints, used as input to our method, are severely inaccurate. However, our method is able to reconstruct correct 3D motion. In the following examples, let *white dancer* and *orange dancer* denote the dancer wearing a white and an orange shirt respectively. Several 2D based skeleton error examples are depicted in the zoomed-in circular insets: (a) Wrong pose estimation of the left arm of the orange dancer; (b) The right arm of the orange dancer is occluded hence detected erroneously; (c) The nose tip of the orange dancer is erroneously detected as the nose tip of the white dancer; (d) Erroneous 2D pose estimation of the white dancer's right hand.

This convolutional layer is used in F_Q only. The second is a multi-head attention layer, used in both F_S and F_Q. A standard attention mechanism looks at the data as a *sequence* of *embeddings*. In our mechanism, the *views* form the sequence, and the *channels* form the embeddings. Our attention layer is based on the LiftFormer [12]. While the LiftFormer attends to time, we attend to views. The embedding size is 1024, and we use 64 heads (see Table 1), so for the Query, Key and Value (each), we have 64 learned linear filters of size $K \times 16$, where K is the number of views and 16 is the result of 1024/64.

A key architectural choice in our fusion layers, F_S and F_Q, is at which stage to fuse the input views. In Figure 5 we depict the conceptual idea of fusing. Each fusing scheme has its own advantages and disadvantages. Following the insight that early convolutional layers yield coarse features and late ones yield semantic features, we observe early fusion as generating all features (coarse and semantic) when a network is aware to all input branches, and observe middle fusion as first generating coarse features that are distinct for each branch, and only then fuse the coarse features together to generate common semantic features. When applying late fusion, the network creates distinct coarse and semantic features

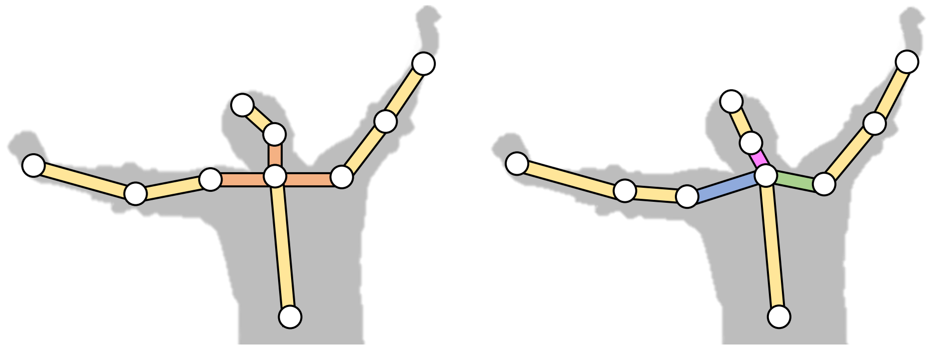

Fig. 4: Skeletal connectivity changes, demonstrated on the neck joint. Left: original connectivity, where shoulders and head are rigidly connected, yielding poor reconstruction. Right: new connectivity, with extra degrees of freedom.

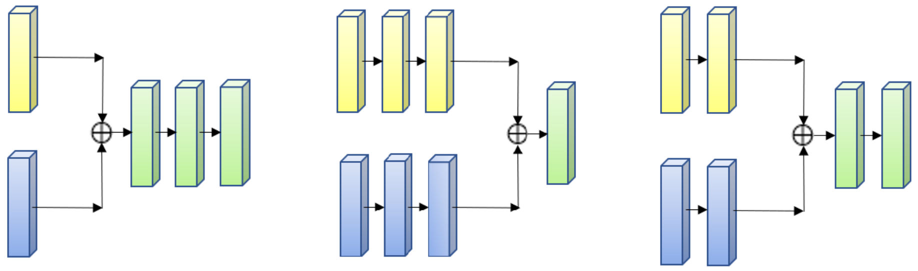

Fig. 5: Fusion schemes. Left: early fusion; Middle: late fusion; Right: middle fusion. Yellow and blue blocks symbolize features from distinct input branches, and green blocks represent fused data. Each block stands for a tensor of features.

for each branch and only then fuses them together. During the development of our model, we have experimented with different fusion schemes, and found out that for our setting the early fusion works best.

Figure 6 depicts diagrams of the multi-view fusion layers and the encoders. The input to both fusion layers is $\mathbf{V}_{s,q,r} \in \mathbb{R}^{T\times 3J\times K}$ (described in the Architecture section of the main paper). In order to make the diagram more intuitive, we sketch V as K temporal sequences. Each temporal sequence is a 2D tensor, where channels are formed by the joints. The fusion layer first streams these temporal sequences through an expansion layer, increasing their channel size. Next, our fusion layer concatenates the expanded data and obtains a 3D tensor, on which it applies multi-view convolutional filters. These filters consider the data from all views. At the next stage we apply a multi-head attention layer that attends to views. Our network uses the attention layer output to create a 2D tensor representing one 'fused' view. The features are then passed to the encoder. The encoder block E_Q consists of three parallel 1D convolutional layers of different kernel sizes, followed by a final additional 1D convolution. The encoder E_S starts with an adaptive max pooling to collapse the time dimension and then runs a final 1D convolution. After each convolution block, we apply

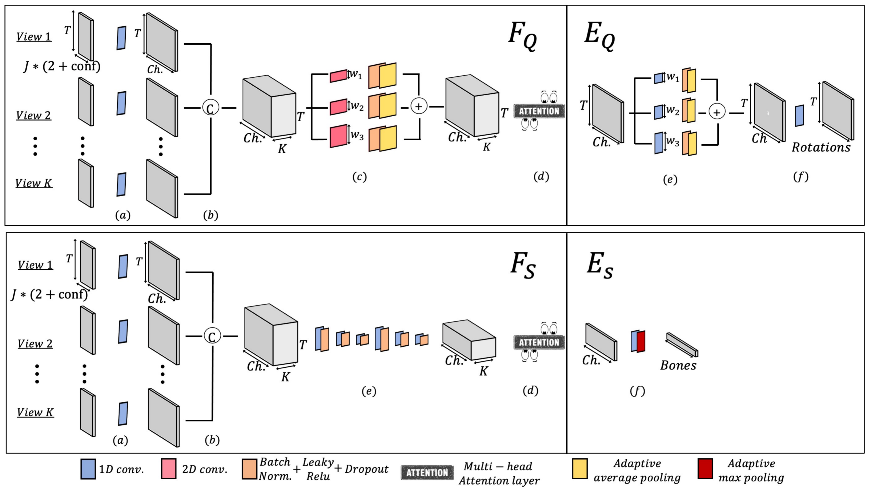

Fig. 6: Architecture in detail. The upper and lower parts are the rotations and bones branches, respectively. (a) Channel-wise expansion layer; (b) View concatenation; (c) Multi-view convolutional filters; (d) Cross-view attention layer (detailed in Figure 7); (e) Single-view convolutional filters; (f) Channel-wise shrinkage layer.

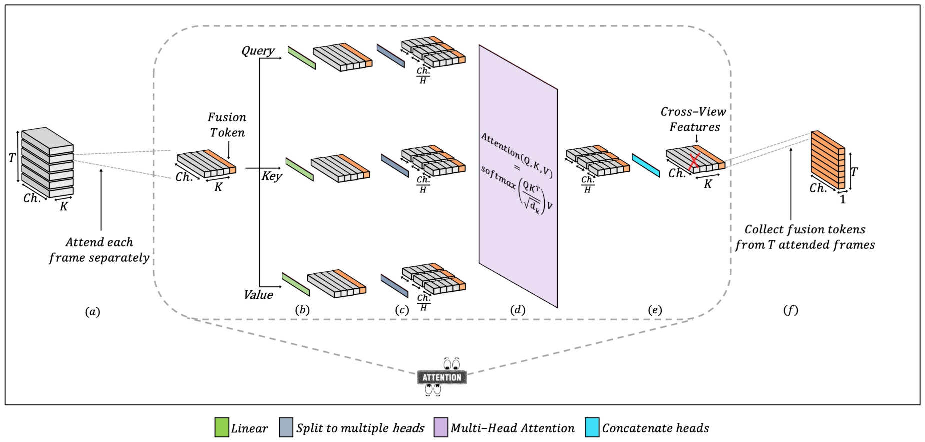

Fig. 7: Cross-view attention layer (Figure 6(d)) in detail. Our attention mechanism processes each frame separately, attending the multi-view features and fusing them to a single output per frame. (a) Process each temporal frame independently. Add a learned token [6] that forms a *fusion view*; (b) Linear layer; (c) Split the channels to H attention heads; (d) Multi-Head attention [16]; (e) Concatenate the attention heads; (f) Drop features from the original views. Collect fusion view features from all the frames in the temporal sequence.

Table 1: FLEX structure. **J** denotes the number of joints, **K** the number of views, and **L** denotes the number of limbs. k and s denote kernel width and the stride, respectively. $\rightarrow$ denotes parallel convolutions while $\downarrow$ denotes sequential ones.

Name		Layers	k	s	in / out
F_Q		1D-Conv + BatchNorm + LReLU + Dropout	1	1	$3J$ / 1024
	$\rightarrow$	2D-Conv + BatchNorm + LReLU + Dropout + Adap AP	5	3	1024 / 1024
	$\rightarrow$	2D-Conv + BatchNorm + LReLU + Dropout + Adap AP	3	1	1024 / 1024
	$\rightarrow$	2D-Conv + BatchNorm + LReLU + Dropout + Adap AP	1	1	1024 / 1024
		Multi-head Attention layer (64 heads)	–	–	1024 / 1024
E_Q	$\rightarrow$	1D-Conv + BatchNorm + LReLU + Dropout + Adap AP	5	3	1024 / 1024
	$\rightarrow$	1D-Conv + BatchNorm + LReLU + Dropout + Adap AP	3	1	1024 / 1024
	$\rightarrow$	1D-Conv + BatchNorm + LReLU + Dropout + Adap AP	1	1	1024 / 1024
		1D-Conv	1	1	$1024/4(J-1)+4K$
D		1D-Conv + ReLU	1	1	$4J$ / 1024
	$\downarrow$	1D-Conv + ReLU + Adap AP	1	1	1024 / $24J$
		Linear	–	–	$24J$ / J
F_S		1D-Conv + BatchNorm + LReLU + Dropout	1	1	$J3$ / 1024
		1D-Conv + BatchNorm + LReLU + Dropout	5	1	1024 / 1024
	$\downarrow$	1D-Conv + BatchNorm + LReLU + Dropout	3	1	1024 / 1024
		1D-Conv + BatchNorm + LReLU + Dropout	1	1	1024 / 1024
		1D-Conv + BatchNorm + LReLU + Dropout	5	1	1024 / 1024
	$\downarrow$	1D-Conv + BatchNorm + LReLU + Dropout	3	1	1024 / 1024
		1D-Conv + BatchNorm + LReLU + Dropout	1	1	1024 / 1024
		Multi-head Attention layer (64 heads)	–	–	1024 / 1024
E_S		Adaptive MP	–	–	–
		1D-Conv	1	1	1024 / L

batch normalization, a leaky rectified linear unit and dropout. Finally, we run a shrinking filter to decrease the number of channels to the desired output size. Table 1 describes the weight parameters of each layer.

In Figure 7 we zoom into the attention block (item (d) in Figure 6). Each slice of one temporal frame is separately forwarded through this block. Such a slice contains features from all the views. Within the attention block, we concatenate an additional view, which we call the *fusion view*. This additional view is a learned token [6], in which the attention mechanism combines significant information from all views. Our model attends to all views, including the added one. After exiting the attention block we omit the data related to the given views and keep the learned fusion view only. This fusion view is then concatenated with the outputs of the other temporal frames.

6 Data

6.1 Train and Evaluation

We train our model on the Human3.6M dataset [8,3]. We evaluate FLEX on the Human3.6M and the KTH Multi-view Football II [11] datasets, and on synthetic multi-person video streams captured by dynamic cameras.

Human3.6M [8,3] is a dataset of 3.6 Million accurate 3D Human poses, acquired by recording the performance of 5 female and 6 male subjects, under 4 different viewpoints. This dataset holds a diverse set of motions and poses encountered as part of 17 typical human activities such as talking on the phone, walking, and eating. As recommended on the Human3.6M dataset page, we use subjects S1, S5, S6, S7, and S8 for training and subjects S9 and S11 for testing. This dataset grants licenses free of charge that are limited to academic use only. More information and access to raw data are provided on the dataset webpage[1].

KTH Multi-view Football II [11] is a dataset of video streams from three synchronized cameras with 800-time frames per camera. The streams depict two different players (in separate streams), where each player has two sequences in varying levels of scene complexity. This dataset is unique in the sense that the cameras are dynamic, hence the approximation of camera extrinsic parameters is very challenging. We adjust the skeleton topology of the KTH dataset to match the topology of Human3.6M in the following way. KTH extracts 14 joints (top-head, mid-head, shoulders, hips, knees, feet, elbows, and hands). We create root (pelvis) and neck joints by averaging the hips and the shoulders respectively and then create a spine joint by averaging the root and the neck. Then we draw bones according to the Human3.6M skeleton topology. The raw data can be accessed and downloaded from the dataset webpage[2]. This dataset may only be used for academic research and not for commercial use.

Ski-Pose PTZ-Camera [14] is a 6 cameras' multi-view pant-tilt-zoom-camera (PTZ) dataset. It features competitive alpine skiers performing giant slalom runs. It holds 20K images, with a single skier in each. The cameras can rotate, but their locations are fixed. This dataset provides labels for the skiers' 3D locations in each frame, their projected 2D locations, and per-frame calibration of the PTZ cameras. In the dataset webpage[3] there are more descriptions of the dataset as well as download instructions.

Our synthetic videos are generated using Mixamo [1] and Blender [5]. We maintain two scenes with two interacting subjects in each. One scene illustrates a boxing arena, and one shows Macarena dancers. We create as many cameras as we wish, with full control on each camera's dynamic motion trajectory. Each synthetic camera outputs a video of the scene, taken from its view. To evaluate FLEX on these videos, we use a network that has been trained on the Human3.6M dataset, introducing satisfactory generalization abilities of our model.

6.2 Input data

The input to our network is 2D joint locations per frame, accompanied by a confidence value. We train our network with several variations of input data.

Ground truth 2D pose Obviously, training with ground truth input data yields the best possible results. We use the 2D labeling provided by the Human3.6M dataset.

[1] `https://vision.imar.ro/human3.6m/`
[2] `https://www.csc.kth.se/cvap/cvg/?page=footballdataset2`
[3] `https://www.epfl.ch/labs/cvlab/data/ski-poseptz-dataset/`

Estimated 2D pose To simulate dynamic capture environments, where 2D labels are not available, we use several state-of-the-art 2D pose estimators as 2D backbones. In our ablation studies we demonstrate the dependency on a good estimator. The estimators that we use are OpenPose [2], CPN [4], and the one used by Iskakov *et al.* [9] (who base their 2D estimation on the "simple baselines" architecture [18]). OpenPose and Iskakov *et al.* provide confidence values that we add to the network input. CPN does not provide these values, hence we assign identical confidence values for all joints when using it. While OpenPose and CPN are dedicated 2D pose estimators, Iskakov *et al.*'s 2D estimator is part of a 3D pose estimator. To extract the 2D pose we retrain their model using its given code and save intermediate values. The 2D pose estimation computed by Iskakov *et al.* [9] uses camera distortion parameters. In addition to being free of extrinsic camera parameters, we are strict about not using the intrinsic ones as well (see Section 3 in the main paper); hence, we retrain Iskakov *et al.* [9] without those parameters.

Skeleton topology may vary between the aforementioned 3D datasets and 2D poses predicted by backbone algorithms. To mitigate this, we make adjustments to the predicted 2D joints. Openpose [2] extract 16 joints (root, neck, mid-head, top-head, shoulders, hips, knees, feet, elbows, and hands). These joints exist in the aforementioned datasets as well. In addition, a spine joint, which exists only in the 3D datasets, is artificially added (calculated as the 2D spatial average between the root and the neck joint). For the CPN [4] 2D prediction, we simply use the values computed by Pavllo *et al.* [13] and provided in their project page, which already possess the requested topology. Lastly, Iskakov *et al.* [9] predict the exact joints required by the aforementioned 3D datasets.

At inference time, when videos from the wild are used, we use a network that was trained using an estimated 2D pose and make sure that during inference, the exact 2D backbone that was used for training, is applied.

6.3 Ground truth

During train time we use 3D joint location ground truth per view, plus rotation ground truth for the discriminator. In contrast to location ground truth, rotation ground truth is required only once, no matter how many views we have. During test time we need none of the above.

7 Camera Parameters

We next formulate the notion of camera parameters. Consider a pinhole camera model. Such a model possesses two types of parameters, extrinsic and intrinsic. *Extrinsic* parameters correspond to

- A rotation matrix R: a matrix of size 3×3 characterizing the rotation from 3D real world axes into 3D camera axes.
- A translation vector T: a vector of size 3 representing the translational offset of the camera in the 3D scene.

Intrinsic parameters, stored in a 3×3 matrix K, are specific to a camera. K consists of the focal length f_x, f_y, the camera optical center c_x, c_y and a skew coefficient s_k:

$$K = \begin{bmatrix} f_x & s_k & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

We denote the mapping from 3D world coordinates into a 2D image plane by a 3×4 matrix P. P is sometimes called *camera matrix* or *projection matrix*. To calculate P, both camera extrinsic and intrinsic parameters are used:

$$P = K \times \left[R \,|\, T\right]. \tag{2}$$

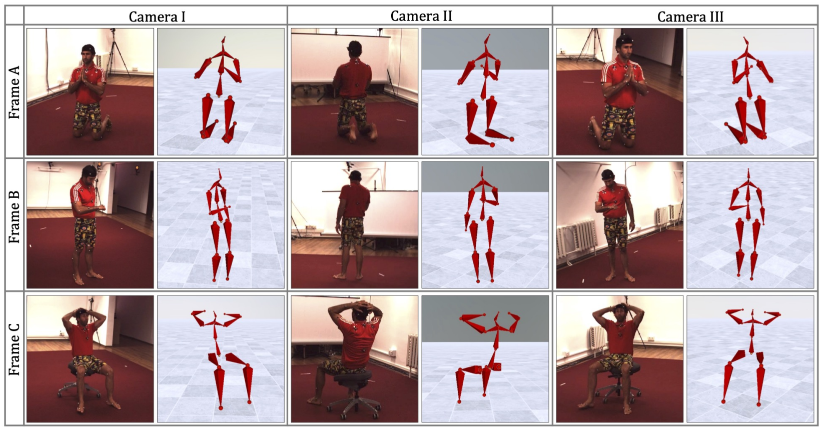

Fig. 8: Additional results on videos from the Human3.6M dataset.

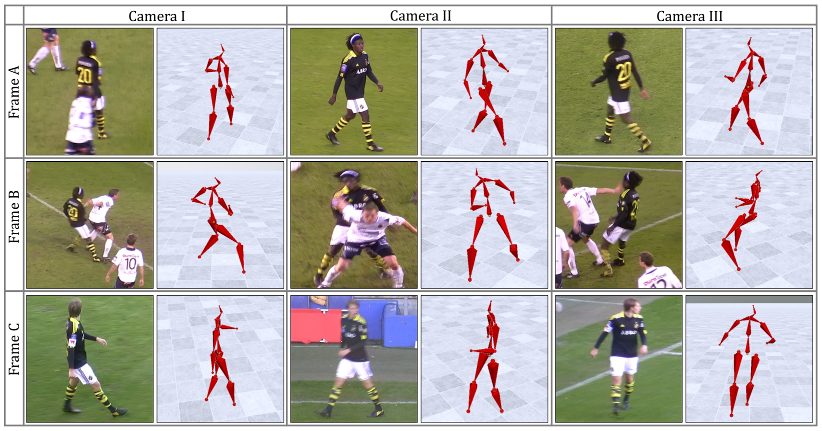

Fig. 9: Additional results on videos from the KTH Multi-view Football II dataset.

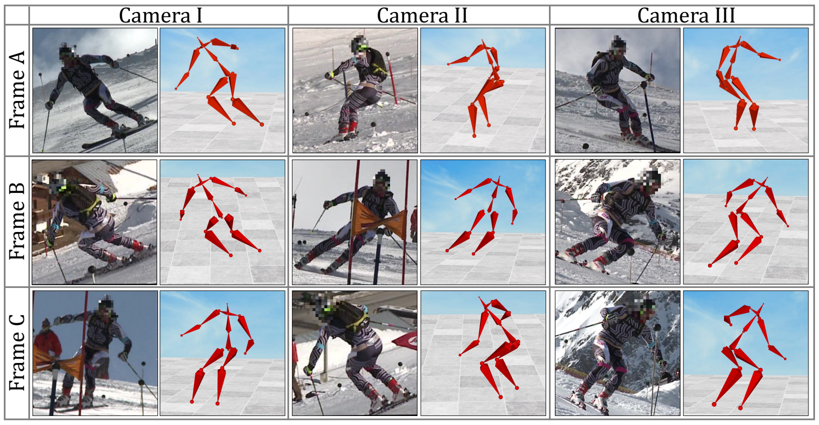

Fig. 10: Enlarged results on the Ski-Pose PTZ-Camera dataset (from main paper).

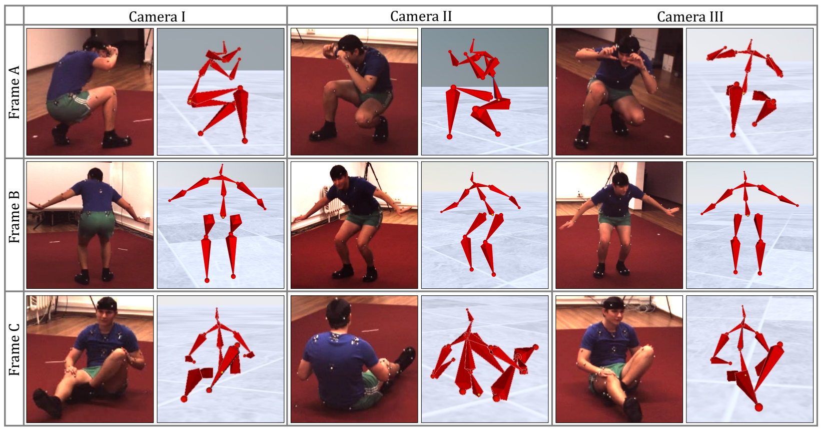

Fig. 11: Enlarged results on the Human3.6M dataset (from main paper).

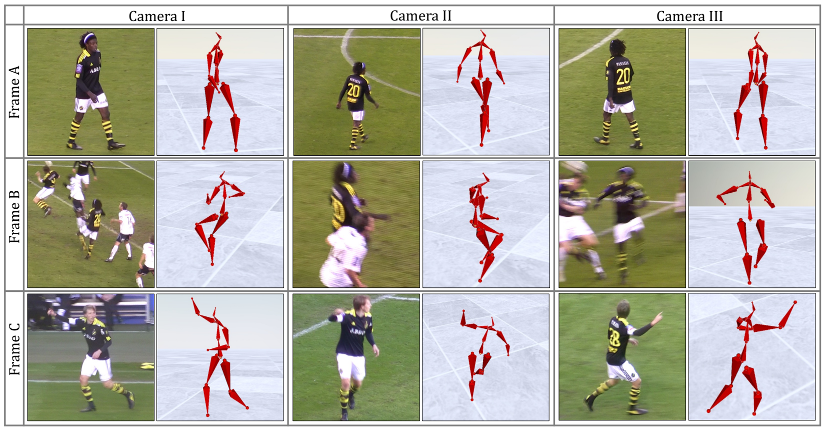

Fig. 12: Enlarged results on the KTH Football II dataset (from main paper).

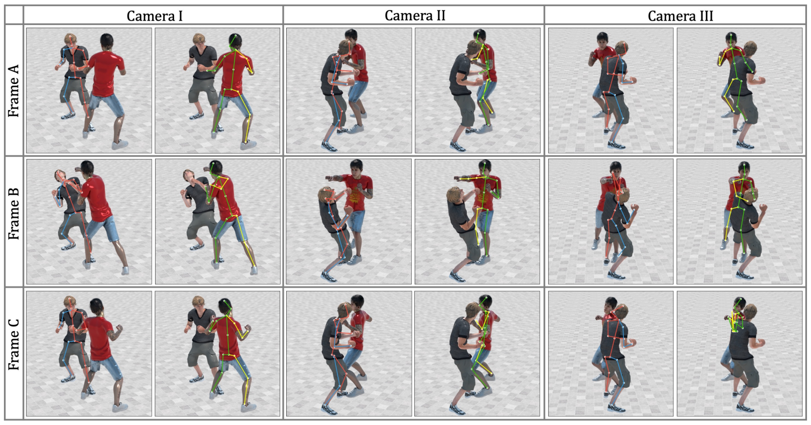

Fig. 13: 2D joint locations estimated on a multi-person synthetic video of boxers.

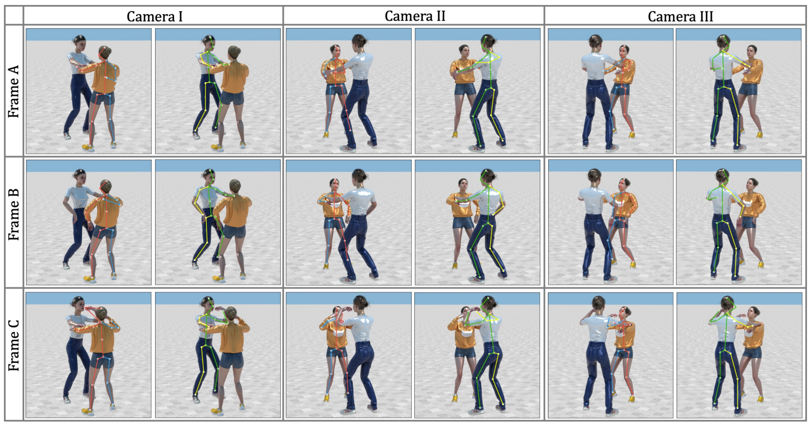

Fig. 14: 2D joint locations estimated on a multi-person synthetic video of dancers.

References

1. Adobe Systems Inc.: Mixamo (2018), `http://www.mixamo.com`
2. Cao, Z., Hidalgo, G., Simon, T., Wei, S., Sheikh, Y.: Openpose: Realtime multi-person 2d pose estimation using part affinity fields. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition. CVPR '18, vol. 43, pp. 172–186. IEEE Computer Society, Washington, DC, USA (2018)
3. Catalin Ionescu, Fuxin Li, C.S.: Latent structured models for human pose estimation. In: International Conference on Computer Vision (2011)
4. Chen, Y., Wang, Z., Peng, Y., Zhang, Z., Yu, G., Sun, J.: Cascaded pyramid network for multi-person pose estimation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7103–7112. IEEE Computer Society, Washington, DC, USA (2018)
5. Community, B.O.: Blender - a 3D modelling and rendering package. Blender Foundation, Stichting Blender Foundation, Amsterdam (2018), `http://www.blender.org`
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2018). https://doi.org/10.48550/ARXIV.1810.04805, `https://arxiv.org/abs/1810.04805`
7. Fang, H.S., Xie, S., Tai, Y.W., Lu, C.: Rmpe: Regional multi-person pose estimation. In: ICCV. p. 2334–2343. IEEE Computer Society, Washington, DC, USA (2017)
8. Ionescu, C., Papava, D., Olaru, V., Sminchisescu, C.: Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. IEEE Transactions on Pattern Analysis and Machine Intelligence (2014)
9. Iskakov, K., Burkov, E., Lempitsky, V.S., Malkov, Y.: Learnable triangulation of human pose. 2019 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 7717–7726 (2019)
10. Kanazawa, A., Black, M.J., Jacobs, D.W., Malik, J.: End-to-end recovery of human shape and pose. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7122–7131. CVPR '18, IEEE Computer Society, Washington, DC, USA (2018). https://doi.org/10.1109/CVPR.2018.00744
11. Kazemi, V., Burenius, M., Azizpour, H., Sullivan, J.: Multi-view body part recognition with random forests. In: BMVC 2013 - Electronic Proceedings of the British Machine Vision Conference 2013. BMVA, UK (09 2013). https://doi.org/10.5244/C.27.48
12. Llopart, A.: Liftformer: 3d human pose estimation using attention models. CoRR **abs/2009.00348** (2020), `https://arxiv.org/abs/2009.00348`
13. Pavllo, D., Feichtenhofer, C., Grangier, D., Auli, M.: 3d human pose estimation in video with temporal convolutions and semi-supervised training. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7753–7762 (2019)
14. Rhodin, H., Spörri, J., Katircioglu, I., Constantin, V., Meyer, F., Müller, E., Salzmann, M., Fua, P.V.: Learning monocular 3d human pose estimation from multi-view images. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition pp. 8437–8446 (2018)
15. Shi, M., Aberman, K., Aristidou, A., Komura, T., Lischinski, D., Cohen-Or, D., Chen, B.: Motionet: 3d human motion reconstruction from monocular video with skeleton consistency. ACM Transactions on Graphics (TOG) **40**(1), 1–15 (2020)

16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017). https://doi.org/10.48550/ARXIV.1706.03762, `https://arxiv.org/abs/1706.03762`
17. Villegas, R., Yang, J., Ceylan, D., Lee, H.: Neural kinematic networks for unsupervised motion retargetting. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8639–8648. CVPR '18, IEEE Computer Society, Washington, DC, USA (2018)
18. Xiao, B., Wu, H., Wei, Y.: Simple baselines for human pose estimation and tracking. In: Proceedings of the European conference on computer vision (ECCV). pp. 466–481. Springer International Publishing, Berlin/Heidelberg, Germany (2018)