Attention-Propagation Network for Egocentric Heatmap to 3D Pose Lifting

Supplementary Material

A. Overview

The supplementary material contains the following:

- Dataset Processing
- Implementation
- Training
- Experiment
- Example Figure
- Limitations and Future Works

B. Dataset Processing

We explain the details of the train and test dataset we used in this section. Our method requires a 2D and 3D pose annotation and stereo input images. The 2D annotation is necessary for generating the heatmaps.

B.1. UnrealEgo

We utilize the full dataset, including metadata files and preprocessed pickles. The public Ego3DPose [5] code loads metadata and pickles. Their code adds 2D and 3D pose data in the camera coordinate system and their limb heatmap representation in the pickle files. Our method uses these final pickles.

B.2. EgoCap

We used publicly available 2D pose annotation on the train set. Additionally, we got the full ground truth 3D pose for the train set of the EgoCap [9] dataset from the authors. In the fisheye views of the dataset, images are projected only in the circular area due to strong distortion. Thus, the original images contain areas that do not have real views. Following the Kang et al. [5], we cropped the image horizontally into a square area centered at the x-axis focal center ($\mathbf{fx}$) provided in the dataset calibration data. We resized the images to 256 by 256 images to fit our model.

The dataset has a train set and 2D and 3D validation sets. The 3D validation set contains a ground truth 3D pose and is used for testing. The 2D validation dataset provides the 2D annotation for the images in the 3D validation sets from a subject labeled 7. The 3D pose is converted from a mm to a cm unit to scale the pose loss in accordance with the UnrealEgo dataset.

C. Implementation

C.1. Grid ViT Heatmap Encoder

The 64×64 sized heatmaps are put into one image with resolution 384×384. The image comprises 36 areas as a 6×6 grid. The number of joint heatmaps is 30 for the UnrealEgo [2] dataset and 34 for the EgoCap [9]. The heatmaps fill in the grid in order. The areas that do not correspond to any heatmap are masked in the ViT encoder module and don't impact the output.

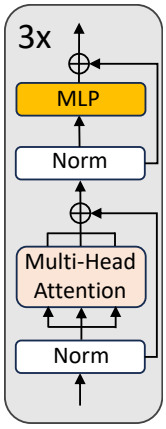


Figure 1. The ViT encoder architecture.

We adopt the ViT encoder [3] architecture. Our implementation adopts the public Transformers [13] module **ViTModel** class for the PyTorch [8]. We removed the $[CLS]$ token since we are not using the module for a classification task. Doing so improves pose estimation accuracy empirically. The module follows the standard ViT [3] encoder architecture shown in Fig. 1 that takes the input embedding z and outputs feature embedding z'.

The ViT encoder takes embeddings of size 1024 per each of $32N_J$ patches, $z = [z_1, z_2, \ldots, z_{32N_J}]$. The multi-head attention layer has 8 heads. The intermediate layer size of the MLP is 4096. The Grid ViT Heatmap Encoder uses three ViT encoder layers. It outputs a total 16384 size of the embedding vector from 16 patches for each heatmap. The embedding vector is then compressed with MLP denoted $E_{\mathbf{K}}$ in the paper. The MLP has ReLU [1] non-linearity for the intermediate layers. The MLP's hidden sizes of the first two layers are 2048 and 512, and the last layer outputs a final embedding of size 128.

C.2. Propagation Network

In an extension of the typical LSTM [4], the Propagation Unit's relational features, joint features, hidden and cell states, and gate outputs all have the same size. We chose 256 for the size.

C.2.1 Limb Heatmap Encoder

The limb heatmap encoder E_R extracts relational features. The encoder consists of three layers with the same structure as the final MLP layers of the Grid ViT Heatmap Encoder, with only an input size difference. The input two-channeled limb heatmap [5] has $2 \times 64 \times 64$ size. The encoder takes it after flattening it. The encoder consists of three fully connected layers, the first two layers with 2048 and 512 output size, with the ReLU [1] activation, and the final layer outputs the embedding with a size 128.

C.2.2 Second Layer of the PU

The second layer of PU does not take distinct relational and joint features. It takes the parent joint's second layer cell and hidden state with the first layer's hidden state of the joint. Since hidden states from different layers are used in this section, let's denote the n-th layer hidden states of i-th joint $h_{n,i}$. The additional forget gate in the second layer g_i controls the parent joint's second PU layer's hidden state, resulting in the modified hidden state $h'_{2,i}$. This is formulated as follows:

$$g_i = \sigma(W_g \cdot h_{1,i} + b_g) \tag{1}$$
$$h'_{2,i} = g_i \odot h_{2,parent(i)} \tag{2}$$

The modified parent hidden state and the joint's first layer hidden state are input for the inner LSTM [4].

C.2.3 Internal LSTM of the PU

We explain the formulation of the LSTM [4] inside the PU in more detail here.

Formulation of typical LSTM. The LSTM is formulated as follows, where h_{i-1} denotes the hidden state of the previous step, c_{i-1} denotes the cell state of the previous step, and x_i denotes the input. Here, W and b denote weights and biases for each gate. The symbol $\odot$ represents element-wise multiplication, and the $+$ sign represents element-wise addition. $\tanh$ and σ denote the hyperbolic tangent and sigmoid activation.

$$f_i = \sigma(W_f \cdot [h_{i-1}, x_i] + b_f) \tag{3}$$
$$i_i = \sigma(W_i \cdot [h_{i-1}, x_i] + b_i) \tag{4}$$
$$o_i = \sigma(W_o \cdot [h_{i-1}, x_i] + b_o) \tag{5}$$
$$\tilde{c}_i = \tanh(W_c \cdot [h_{i-1}, x_i] + b_c) \tag{6}$$
$$c_i = f_i \odot c_{i-1} + i_i \odot \tilde{c}_i \tag{7}$$
$$h_i = o_i \odot \tanh(c_i) \tag{8}$$

The f_i, i_i, and o_i are forget, input, and output gates. $\tilde{c}_i$ denotes the candidate cell value. h_i and c_i are the final hidden and cell state for step i.

Formulation of internal LSTM. Unlike the LSTM taking the cell and hidden state, the internal LSTM of the first PU layer takes three states in addition to input joint features. The three states are the modified parent's hidden state h'_i, the modified relational feature of the joint r'_i, and the cell state of the parent $c_{parent(i)}$. The input is joint features $\mathbf{F_{J,i}}$.

This section explains the first and second layers together; thus, we denote the n-th layer of i-th joint with a n, i subscript, as in $h_{n,i}$ for the hidden state. In the computation of the forget, input, and output gates and the candidate cell value, a concatenated vector of the modified parent's hidden state and relational features, and the joint features $[h'_{1,i}, r'_{1,i}, \mathbf{F_{J,i}}]$ replaces $[h_{i-1}, x_i]$.

$$f_{1,i} = \sigma(W_{1,f} \cdot [h'_{1,i}, r'_i, \mathbf{F_{J,i}}] + b_{1,f}) \tag{9}$$
$$i_{1,i} = \sigma(W_{1,i} \cdot [h'_{1,i}, r'_i, \mathbf{F_{J,i}}] + b_{1,i}) \tag{10}$$
$$o_{1,i} = \sigma(W_{1,o} \cdot [h'_{1,i}, r'_i, \mathbf{F_{J,i}}] + b_{1,o}) \tag{11}$$
$$\tilde{c}_{1,i} = \tanh(W_{1,c} \cdot [h'_{1,i}, r'_i, \mathbf{F_{J,i}}] + b_{1,c}) \tag{12}$$

For the second layer, the modified second layer parent hidden state $h'_{2,i}$ from the Sec. C.2.2 takes the place of h_{i-1}. The previous layer's hidden state $h_{1,i}$ replaces input x_i, analogous to the standard multi-layered LSTM.

$$f_{2,i} = \sigma(W_{2,f} \cdot [h'_{2,i}, h_{1,i}] + b_{2,f}) \tag{13}$$
$$i_{2,i} = \sigma(W_{2,i} \cdot [h'_{2,i}, h_{1,i}] + b_{2,i}) \tag{14}$$
$$o_{2,i} = \sigma(W_{2,o} \cdot [h'_{2,i}, h_{1,i}] + b_{2,o}) \tag{15}$$
$$\tilde{c}_{2,i} = \tanh(W_{2,c} \cdot [h'_{2,i}, h_{1,i}] + b_{2,c}) \tag{16}$$

The Propagation Unit takes features from the parent joint, not the previous index. In the computation of the final cell and hidden state, both layers of PU take $c_{n,parent(i)}$ instead of c_{i-1} in the formula. The hidden state is computed in the same way.

$$c_{n,i} = f_{n,i} \odot c_{n,parent(i)} + i_{n,i} \odot \tilde{c}_{n,i} \tag{17}$$
$$h_{n,i} = o_{n,i} \odot \tanh(c_{n,i}) \tag{18}$$

D. Training

D.1. Hardware Setup

We trained and tested our method on a server with NVIDIA RTX A6000 GPU and AMD EPYC 7313 16-Core Processor CPU.

D.2. Heatmap Estimator

The heatmap estimator is trained using UnrealEgo [2] code and their scripts for the UnrealEgo dataset. The default configuration utilizes Adam [6] optimizer with a learning rate 10^{-3}. They train the network for 10 epochs, the later

5 epochs with linear decay, with batch size 16. For the EgoCap dataset, we trained the heatmap estimators for 30 epochs with the same setup. Linear decay is used for the last 15 epochs proportionally.

When hasty convergence, where all heatmap values converge to 0, is detected, the training is automatically restarted, following the protocol of Kang et al. [5].

D.3. EgoTAP Network

The network is trained with the AdamW [7] optimizer with pretrained and frozen heatmap estimator weight. The learning rate of 10^{-3} is used with 16 epochs with a cosine annealing scheduler, with batch size 32. Early epochs use a linear warmup, one epoch for the UnrealEgo [2], and two epochs for the EgoCap [9] dataset.

D.4. Loss

D.4.1 EgoTAP Network

EgoTAP has two loss terms: a pose error loss (i.e., joints' average Euclidean distance) and cosine-similarity loss [11] that focuses on estimating the correct 3D orientation for each limb.

The pose error loss is defined as follows. Given two 3D joint poses: the predicted pose $\mathbf{p}\prime_i$ and the ground truth pose $\mathbf{p}_i$, for $i = 1, \ldots, J$, where J is the total number of joints:

$$L_p = \frac{1}{J}\sum_{i=1}^{J} \|\mathbf{p}\prime_i - \mathbf{p}_i\|_2 \tag{19}$$

The cosine similarity loss is then defined as follows. A limb pose vector for a particular joint is obtained by subtracting the pose of its parent joint from its pose, i.e., $\mathbf{v}_i = \mathbf{p}_i - \mathbf{p}_{\text{parent}(i)}$ and $\mathbf{v}\prime_i = \mathbf{p}\prime_i - \mathbf{p}\prime_{\text{parent}(i)}$ for the ground truth and predicted poses, respectively. Given these vectors, the cosine similarity between two limb pose vectors is calculated using their inner product:

$$L_c = \frac{1}{J-1}\sum_{i=2}^{J} \| \frac{\mathbf{v}_i \cdot \mathbf{v}\prime_i}{\|\mathbf{v}_i\|_2 \|\mathbf{v}\prime_i\|_2} \tag{20}$$

Note that the root joint is ignored since it does not have a parent joint.

The final loss term is a weighted sum of two losses, where we choose $w_p = 0.1$ and $w_c = -0.01$. The cosine similarity loss weight has a negative sign because higher cosine similarity indicates a more accurate pose.

$$L = w_p L_p + w_c L_c \tag{21}$$

D.4.2 Heatmap Reconstruction Network for Ablation

We adopted the heatmap decoder proposed by Tome et al. [11] for the heatmap reconstruction in the ablation study of the Grid ViT Heatmp Encoder. The network with only mean squared loss struggles from early convergence to outputting near-zero valued heatmaps. The problem is more severe than the heatmap estimator network since the reconstruction network does not contain specialized architecture like the U-Net [10], which helps the heatmap generation through the multi-resolution features.

Thus, additional loss to match the heatmap's minimum and maximum values is added if the mean squared loss is higher than the threshold to prevent the network training in the ablation studies from converging to outputting only zeros. We set the threshold to a value empirically found sufficient to ensure avoidance of the zero-only convergence. The loss term guides the network to output a peak in the heatmap, as joint heatmaps do.

The heatmap reconstruction's target is to minimize the mean squared loss between the predicted heatmap H, and reconstructed heatmap $H\prime$. This is computed as follows:

$$L_r = \frac{1}{N}\sum_{i=1}^{N}(H_i - H'_i)^2 \tag{22}$$

The min-max loss applies only if the mean squared loss is higher than threshold θ. The threshold is $5.5 * 10^{-4}$. It is computed as follows:

$$L_{min} = \frac{1}{N}\sum_{i=1}^{N} |\min(H_i) - \min(H'_i)| \tag{23}$$

$$L_{max} = \frac{1}{N}\sum_{i=1}^{N} |\max(H_i) - \max(H'_i)| \tag{24}$$

$$L_m = \begin{cases} L_{min} + L_{max} & \text{if } L_r > \theta \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

The weight for the reconstruction w_r is set to 1 and the weight for the min-max penalty w_m is set to 10^{-3}, resulting in the total loss:

$$L = w_r \cdot (L_r + w_m \cdot L_m) \tag{26}$$

The loss term does not impact the final result once the network avoids the early convergence and stabilizes. The zero output convergence is still observed for the CNN encoder embedding, so the training was restarted until it did not converge to output only zeros. Such an additional loss term is necessary to get meaningful non-zero reconstruction from the output of the CNN heatmap encoder, showing the difficulty of heatmap information recovery from its embeddings.

MPJPE (PA-MPJPE)								
Method	Jumping	Falling Down	Exercising	Pulling	Singing	Rolling	Crawling	Laying
EgoGlass [15]	78.93(63.85)	123.80(92.71)	94.21(69.85)	79.41(55.41)	68.16(50.25)	100.53(87.26)	173.69(111.51)	106.41(86.42)
UnrealEgo [2]	61.66(49.46)	108.73(78.02)	77.14(58.87)	57.01(43.51)	52.61(37.58)	73.38(64.56)	162.90(102.15)	82.60(67.47)
Ego3DPose [5]	52.12(43.29)	86.08(71.72)	67.52(56.39)	48.92(37.02)	43.86(34.54)	74.24(64.81)	138.47(92.93)	78.13(67.23)
Ours	**43.05(37.31)**	**75.77(63.48)**	**52.76(46.21)**	**34.45(26.82)**	**33.96(29.10)**	**52.24(47.58)**	**126.23(91.46)**	**66.38(59.56)**
Method	Sitting on the Ground	Crouching	Crouching and Turning	Crouching to Standing	Crouching-Forward	Crouching-Backward	Crouching-Sideways	Standing-Whole Body
EgoGlass	204.35(147.99)	121.76(100.12)	130.24(104.28)	84.31(59.67)	82.84(66.06)	90.36(76.83)	101.37(78.81)	69.78(52.59)
UnrealEgo	190.26(144.27)	96.69(79.29)	116.59(99.94)	66.20(44.92)	56.10(46.62)	62.54(46.21)	72.35(55.87)	52.91(39.14)
Ego3DPose	143.44(122.93)	82.01(67.94)	104.24(83.98)	58.74(41.34)	48.60(38.81)	47.36(36.05)	57.85(48.83)	45.69(35.47)
Ours	**121.24(103.27)**	**67.27(60.07)**	**89.97(70.78)**	**41.91(28.68)**	**33.47(29.52)**	**34.08(28.30)**	**40.21(38.51)**	**32.77(28.27)**
Method	Standing-Upper Body	Standing-Turning	Standing to Crouching	Standing-Forward	Standing-Backward	Standing-Sideways	Dancing	Boxing
EgoGlass	69.24(49.36)	77.77(60.27)	83.86(81.65)	76.75(63.23)	78.40(59.83)	82.71(66.46)	82.84(65.59)	66.98(49.13)
UnrealEgo	50.97(34.86)	60.42(46.27)	48.09(40.5)	56.23(47.90)	57.14(44.90)	63.10(50.86)	64.73(51.79)	52.13(38.36)
Ego3DPose	44.14(32.99)	51.45(41.91)	55.66(45.08)	49.48(44.65)	45.35(36.52)	52.25(44.97)	55.30(46.47)	41.55(32.14)
Ours	**31.29(25.51)**	**41.24(35.55)**	**37.07(30.28)**	**39.64(38.06)**	**32.43(30.25)**	**39.34(37.94)**	**42.14(38.39)**	**29.97(25.69)**
Method	Wrestling	Soccer	Baseball	Basketball	American Football	Golf		
EgoGlass	84.23(62.81)	81.57(60.59)	76.20(56.11)	78.33(57.30)	102.54(84.03)	69.69(48.15)		
UnrealEgo	67.85(52.73)	67.09(51.43)	62.15(48.60)	64.73(47.79)	89.57(68.49)	55.87(40.34)		
Ego3DPose	57.96(45.94)	59.56(45.23)	56.21(42.17)	56.02(41.94)	77.89(62.56)	48.10(36.01)		
Ours	**44.15(39.38)**	**48.27(38.56)**	**44.83(35.92)**	**45.19(36.78)**	**65.30(54.41)**	**38.97(31.25)**		

Table 1. Quantitative evaluation results on the **UnrealEgo** dataset per category.

	UnrealEgo [2]	EgoCap [9]
Estimated Heatmap	41.06	55.38
Ground Truth Heatmap	6.63	26.63

Table 2. Comparison of pose estimation error (MPJPE) of our method, with estimated and ground truth heatmaps provided as input. Columns indicate two datasets.

E. Experiment

F. Categorical Evaluation on the UnrealEgo dataset

Table 1 categorically shows the result on the UnrealEgo [2] dataset. Metrics from all three baseline methods, EgoGlass [15], UnrealEgo [2], and Ego3DPose [5] are shown with our method. The MPJPE values are outside the bracket, and the PA-MPJPE values are inside the bracket.

F.1. Per Joint Error Distribution

Fig. 2 shows the CDF (Cumulative Distribution Function) of the error of each joint. Two results show one from the UnrealEgo [2] method as an example of the baseline in the introduction and one for our method, both evaluated on the UnrealEgo dataset.

The thigh is directly attached to the pelvis, which is the origin of the local pose definition in the dataset. Thus, for all methods, the thigh has the lowest errors among all joints. Lower body joints, calf, foot, and balls generally have significantly higher errors than other joints, except for the hands, which have larger errors than the calf. The error of the hand and the lower arm gets much closer to the upper arm in our method, showing the benefit of the propagation.

F.2. Impact of the Heatmap Estimation Accuracy

We experimented with our architecture's performance when the ground truth heatmaps were provided instead of the estimated heatmaps. Table 2 reveals that the limited 2D pose information from the view is a key bottleneck for egocentric pose estimation. The full 2D pose provided by the ground truth heatmap reduces error significantly. Despite the better view provided by the camera attached far from the head, the EgoCap has a higher estimation error with ground truth heatmaps. A relatively small dataset volume for training can also be a bottleneck.

F.3. Impact of ViT Backbone Size

Experiments revealed that the bottleneck of the pose estimation accuracy is not in the computational capacity of the backbone. We experimented with up to 12 layers of the ViT encoders and 8 times larger feature sizes in the ViT encoder. No notable improvement was observed compared to the smaller backbone we chose. The UnrealEgo [2] shows consistent experimental results that the larger ResNet backbones do not improve the pose estimation accuracy.

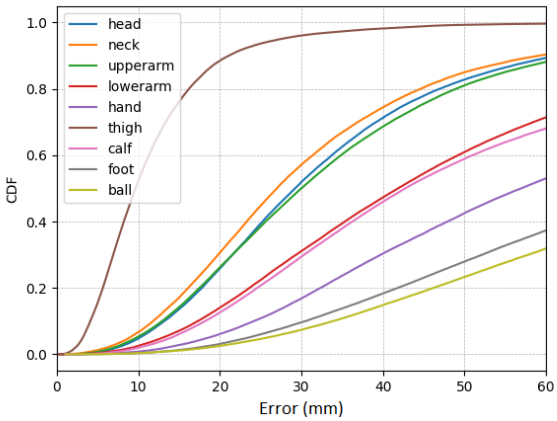
G. Example Figure

G.1. Limb Heatmaps

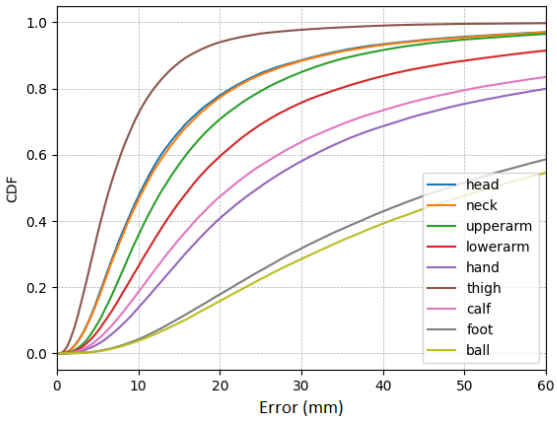
The main text mentions that the limb heatmap estimation is less accurate on the EgoCap [9]. The heatmap visualization in Fig. 3 shows noisy lines for limbs.

H. Limitations and Future Works

EgoTAP is limited to a single frame input and relies fully on visual cues. The result with the EgoTAP on motions with severe occlusion, such as "Crawling" and "Sitting on the Ground", has very high error compared to other motion categories as shown in Table 1. Unlike many recently



(a) UnrealEgo [2]



(b) Ours

Figure 2. CDF of errors for each joint in the UnrealEgo [2] dataset with their method.

(a) EgoCap [9] dataset (b) UnrealEgo [2] dataset

Figure 3. Estimated limb heatmaps on the test set of the **EgoCap** (Left) and **UnrealEgo** (Right).

proposed general pose estimation methods, the egocentric setup's exploration of utilizing the temporal context is limited. For the egocentric view with a limited view, the invisible joints' pose can benefit significantly from the temporal context. For one example in the egocentric setup, Wang. et al. [12] applied temporal optimization using a variational autoencoder for improved pose estimation in the global coordinate.

The method's applicability can further be tested on monocular and different potential egocentric camera setups. The Propagation Network is based on the stereo setup, which provides sufficient information for a 3D pose when the joint is visible from both views. Thus, the propagation scheme helps estimate the child's joint pose. While the 3D pose estimation from the single heatmap is not feasible in the monocular setup, pose space is highly constrained, and our method can also be applicable potentially with modification.

The Propagation Network applies to an egocentric view with a specific characteristic. The method itself lacks dynamicity like the GCN-based method [14], which would make it applicable to many different situations. The tree hierarchy assumption still holds for arbitrary root joints in the skeletal hierarchy, giving room for more dynamicity. Applying such a tree hierarchy-based network has the potential for a specific joint-related situation, such as collision. Such application remains a future work.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. cite arxiv:1803.08375Comment: 7 pages, 11 figures, 9 tables. 1, 2

[2] Hiroyasu Akada, Jian Wang, Soshi Shimada, Masaki Takahashi, Christian Theobalt, and Vladislav Golyanik. Unrealego: A new dataset for robust egocentric 3d human motion capture. In *European Conference on Computer Vision (ECCV)*, 2022. 1, 2, 3, 4, 5

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021. 1

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 1997. 1, 2

[5] Taeho Kang, Kyungjin Lee, Jinrui Zhang, and Youngki Lee. Ego3dpose: Capturing 3d cues from binocular egocentric views. In *SIGGRAPH Asia 2023 Conference Papers*, New York, NY, USA, 2023. Association for Computing Machinery. 1, 2, 3, 4

[6] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 2

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 3

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch:

An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 1

[9] Helge Rhodin, Christian Richardt, Dan Casas, Eldar Insafutdinov, Mohammad Shafiei Rezvani Nezhad, Hans-Peter Seidel, Bernt Schiele, and Christian Theobalt. Egocap: Egocentric marker-less motion capture with two fisheye cameras. *ACM Transactions on Graphics*, 35, 2016. 1, 3, 4, 5

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. cite arxiv:1505.04597Comment: conditionally accepted at MICCAI 2015. 3

[11] Denis Tome, Patrick Peluse, Lourdes Agapito, and Hernan Badino. xr-egopose: Egocentric 3d human pose from an hmd camera. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7728–7738, 2019. 3

[12] Jian Wang, Lingjie Liu, Weipeng Xu, Kripasindhu Sarkar, and Christian Theobalt. Estimating egocentric 3d human pose in global space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11500–11509, 2021. 5

[13] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. 1

[14] Ailing Zeng, Xiao Sun, Lei Yang, Nanxuan Zhao, Minhao Liu, and Qiang Xu. Learning skeletal graph neural networks for hard 3d pose estimation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11416–11425, 2021. 5

[15] Dongxu Zhao, Zhen Wei, Jisan Mahmud, and Jan-Michael Frahm. Egoglass: Egocentric-view human pose estimation from an eyeglass frame. In *2021 International Conference on 3D Vision (3DV)*, pages 32–41, 2021. 4