Mocap Everyone Everywhere: Lightweight Motion Capture With Smartwatches and a Head-Mounted Camera (Supplementary Material)

Jiye Lee
Seoul National University
kay2353@snu.ac.kr

Hanbyul Joo
Seoul National University
hbjoo@snu.ac.kr

1. Supplementary Video

The supplementary video in the project page shows the real-world demonstrations of our motion capture system with two smartwatches and a head-mounted camera. The real-world demonstrations include various scenarios including expansive outdoor scenes, everyday motions and interactions (e.g., making coffee), motions with dynamic movements, and social interactions among multiple people. We also visually demonstrate floor level changes in areas with drastic floor level changes such as walking down the stairs.

We demonstrate comparison with previous state-of-the-art methods that estimates motion from full-body IMU sensor setups [13, 24]. The video also includes several ablation studies of our method, showing the contribution of (1) floor level update module on non-flat scenes in module $\mathcal{F}_{est}$ (Sec. 3.3) (2) motion optimization $\mathcal{F}_{opt}$ with visual cues ϕ_E and ϕ_T (Sec. 3.4).

2. Additional Experiments

2.1. Camera-Derived Head Poses

In the main paper, we have demonstrated the strength of our motion estimation module $\mathcal{F}_{est}$ against baselines, where we test with the head pose generated by available motion capture data given that paired egocentric videos are not available in the existing real IMU dataset. As an extension of this experiment, we further perform a similar quantitative evaluation with egocentric videos using the dataset in EgoLocate [23], where the motions in the TotalCapture [21] dataset are paired with egocentric videos taken from virtual head-mounted cameras in synthetic scenes. This additional experiment provides further evidence of the strength of our method in a closer scenario with the practical setup, accounting for potential noise introduced during the head pose estimation process in monocular SLAM. To demonstrate the reliability of the pre-processing stage of deriving head pose from egocentric video input, which happens prior to $\mathcal{F}_{est}$, we use the dataset in Egolocate [23] where the motions in the TotalCapture [21] dataset are paired with egocentric videos taken from virtual head-mounted cameras in synthetic scenes.

We randomly select 30 sequences in total (about 60000 frames), 15 for each scene. We follow the procedure in Sec 3.2 to obtain head poses from the egocentric videos using DROID-SLAM [20]. As the dataset does not undergo the alignment procedure in our system beforehand (Supp. Sec. 3), the camera poses obtained from SLAM are aligned to world coordinates using head poses derived from the dataset.

We first demonstrate the quality of the the camera-driven head poses from SLAM in Table 1, where it shows minor differences from the head poses of the ground truth motion capture data; rotation error within 2-3 degrees and position error within 5 centimeters. This result can support the validity of our quantitative evaluations in our main paper.

In Table 2, we compare the motion estimation outputs with the camera-driven head poses, denoted as $\mathbf{H}^C + \mathcal{F}_{est}$, with the results with the methods with 6 IMU sensors, PIP [24] and TIP [13], and our method $\mathcal{F}_{est}$ with head poses from the dataset. Our results with camera-driven head pose from the raw egocentric data mostly outperform the competing methods which use the 6 IMU sensors, even though our setup with an egocentric camera and fewer sensors (2 IMUs on the wrists) is much more challenging compared to the previous methods.

We furthermore compare our results with EgoLocate [23], which leverages 6 IMU sensors on the full body for body pose estimation and an additional head-mounted camera for global translation localization. Interestingly, despite the reduced number of sensors, our results outperforms EgoLocate. Especially for root-related position error terms (MPJPE, Root PE) our method significantly outperforms. Such improvement, despite of reduced number of sensors, initially stems from the accuracy of camera pose estimation from DROID-SLAM [20] (Table 1), and by directly incorporating the 6DoF head pose cues from the cameras to the

Scene	Rot. Error (deg)	Pos. Error (cm)	Frames
`flood-ground`	2.69	5.09	27026
`japan-office`	2.12	4.73	32421

Table 1. Rotation and position error of camera-derived head poses compared to ground truth. (Frames are counted in 30 FPS)

Scene	Method	r.MPJPE	Root PE	MPJPE
`flood-ground`	PIP [24]	4.76	40.35	40.81
	TIP [13]	5.13	42.06	41.53
	EgoLocate [23]	4.93	30.87	31.32
	$\mathbf{H}^C + \mathcal{F}_{est}$	4.76	10.52	11.44
	$\mathcal{F}_{est}$	**4.64**	**4.21**	**5.12**
`japan-office`	PIP [24]	4.34	27.20	27.76
	TIP [13]	4.75	30.95	31.63
	EgoLocate [23]	4.43	23.35	23.72
	$\mathbf{H}^C + \mathcal{F}_{est}$	4.37	9.76	10.64
	$\mathcal{F}_{est}$	**4.13**	**3.75**	**4.58**

Table 2. Comparison on motion estimation results with camera-derived head pose as input. $\mathbf{H}^C$ indicates head poses from camera trajectory of egocentric video $\mathbf{I}$. All metrics are in cm scale.

motion estimation process, unlike EgoLocate that employs a decoupled approach between motion estimation (IMU) and localization (camera).

The discrepancy in the results of the root-included metrics between $\mathbf{H}^C + \mathcal{F}_{est}$ and $\mathcal{F}_{est}$ can be considered due to the camera position errors, as depicted in Table 1.

2.2. Ablation Studies on Motion Estimation

Although the motion estimation module $\mathcal{F}_{est}$ does not take the absolute position of wrists as input, our module $\mathcal{F}_{est}$ shows comparable or improved results compared to VR based baselines. (Table 2 of the main paper) To further clarify such result, we also conduct an ablation study on the motion estimation module to demonstrate the contribution of components in $\mathcal{F}_{est}$ for the performance. We first show the contribution of the multi-stage architecture (Sec. 3.3) where the initial module, $\mathcal{F}^{ee}$, determines end-effector positions before estimating the whole body motion. Moreover, we highlight the contribution of data representation in input which combines normalized and global coordinates. This combination of coordinates is advantageous as normalization standardizes actions and removes unwanted variations, while global information resolves ambiguities, like distinguishing between standing and sitting still. This dual-coordinate approach facilitates easier learning and enhances the network's robustness in making inferences. Results in Table 3 validate the contribution of each component.

3. System Setup

3.1. Sensors

Readings IMU Sensor Signals from Smartwatches. For demonstrations with smartwatches, we use Apple Watch SE2 [1] and the SensorLog app [4] to record and access IMU sensor data. From the recorded sensor signals, IMU rotations are obtained from "motionQuaternion" and accelerations from "motionAcceleration" of the Apple Watch OS [2]. The sensor data are recorded in 30 FPS. Furthermore, we apply an average filter on the acceleration data (window length: 7). As previously demonstrated [13], applying the average filter on both real and synthetic acceleration data makes the data sufficiently similar to each other. Note that our module is trained on synthetic IMU data, and we apply our trained model on real IMU data from smartwatches without additional fine-tuning, different from the previous methods [24, 25].

Head-Mounted Camera. We use GoPro cameras for the head-mounted camera. To determine the timecode (in microseconds) when the camera shutter is pressed, we use the Open-GoPro Python SDK [3] to activate the camera shutter.

data rep	multi-stage	MPJPE(↓)	r.MPJPE(↓)	MPJVE(↓)
		8.50	5.85	27.63
✓		6.71	5.58	23.69
✓	✓	**5.20**	**4.95**	**17.00**

Table 3. Ablation studies on the components of $\mathcal{F}_{est}$. Settings are identical as in Tab. 1 (AMASS) in the main paper.

3.2. Calibration and Alignment

For coordinate alignment, the user has to follow 4 steps: (1) put the smartwatches (IMU sensors) on the calibration board for 3 seconds as in Fig. 1 (a); (2) wear the watches and stand in T-pose for 3 seconds; (3) put the camera on 4 positions on the calibration board as in Fig. 1 (b); (4) stand still for 10 seconds and swing arms up and down as in Fig. 4 for time synchronization.

IMU Sensor Calibration. In sensor data readings in real IMU sensors (e.g., smartwatches), the IMU sensor data should first be calibrated to be coherent with the real-world coordinates where the gravity direction is in the negative z-axis. Here the y-axis is defined as the user's facing direction. We use the notations similar to TIP [13], which are as follows:

$$\begin{aligned} \mathbf{R}_g^j &= \mathbf{R}_g^r \mathbf{R}_r^S \mathbf{R}_S^j \\ \mathbf{a}_g &= \mathbf{R}_g^r \mathbf{R}_r^S \mathbf{a}_r \end{aligned} \tag{1}$$

Let the reference coordinate where raw IMU sensors signals are defined as r, and the global coordinate coherent with real-world coordinates as g. Finally, as the module $\mathcal{F}_{est}$ expects wrist joint rotations, the rotation value of IMU sensors should be calibrated into wrist joint rotations $\mathbf{R}_g^j$. Similarly, the raw acceleration data are defined in the reference coordinate of sensors and should be converted into coordinate g. Fig. 3 includes visualization of raw and calibrated signals.

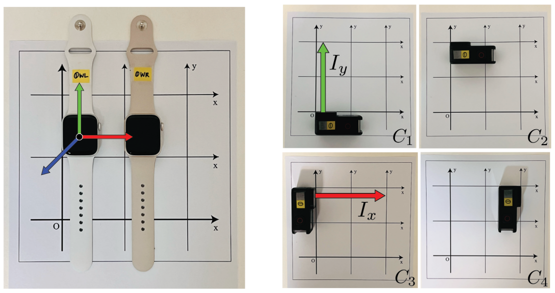
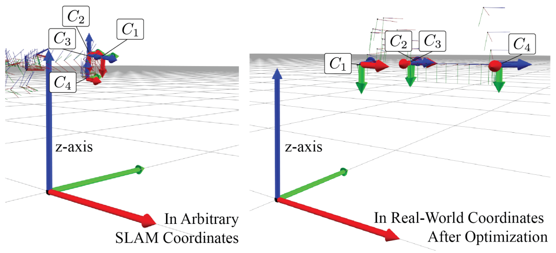


Figure 1. (a) Placing IMU sensors for IMU sensor calibration. The negative z-axis of the sensors are aligned to the gravity direction, and the y-axis is set as a "facing direction". (b) Placing cameras on axes I_x and I_y for camera coordinate alignment.



C_1
C_2
C_3
C_4
z-axis
In Arbitrary SLAM Coordinates
C_1
C_2
C_3
C_4
z-axis
In Real-World Coordinates After Optimization

Figure 2. (a) Camera poses C_i in arbitrary SLAM coordinates. (b) Camera poses C_i in real-world coordinates after alignment optimization.

$\mathbf{R}_r^S$ indicate rotations of raw IMU sensor data defined in the initial reference frame. At the first calibration stage, the user is asked to align the smartwatches to the specified coordinate g (Fig. 1 (a)) so that in the moment $\mathbf{R}_r^S = \mathbf{R}_r^g$. From $\mathbf{R}_r^g$ we can derive $\mathbf{R}_g^r$ by applying $(\mathbf{R}_r^g)^{-1}$. Following previous methods [13, 25], the sensors are placed still for 3 seconds and the signal is averaged. Provided that the axes are aligned, it is not necessary for the two IMU sensors to be in the exact same position as IMU sensor signals are translation invariant.

The next calibration step is to obtain wrist rotations $\mathbf{R}_g^j$ from $\mathbf{R}_g^S$. This is done by applying $\mathbf{R}_S^j$, or $\mathbf{R}_g^S\mathbf{R}_S^j$. To obtain $\mathbf{R}_S^j$, the user wears smartwatches on both wrists and stands in T-pose, facing the y-axis in g.

$$\mathbf{R}_{S_T}^{j_T} = (\mathbf{R}_r^{S_T})^{-1}\mathbf{R}_r^g\mathbf{R}_g^{j_T} \tag{2}$$

The subscript T indicates the values obtained during T-pose calibration. Assuming that the IMU sensors are fixed in the wrist position during capture, we can assume that $\mathbf{R}_S^j$ is fixed, or $\mathbf{R}_{S_T}^{j_T} = \mathbf{R}_S^j$. As the joint rotations in T-pose, or $\mathbf{R}_g^{j_T}$, is known, $\mathbf{R}_{S_T}^{j_T}$ can be derived by obtaining raw sensor recordings during T-pose calibration $\mathbf{R}_r^{S_T}$. Similar to the first step, the user stands in T-pose for 3 seconds.

Note that unlike previous methods [13, 25] we do not consider acceleration bias (mostly, gravity) as the Apple

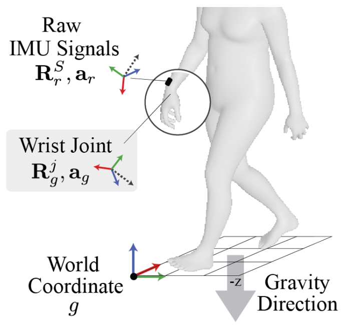


Figure 3. Visualization of the raw and calibrated IMU signals defined in reference coordinate r and user-specified global coordinate g.

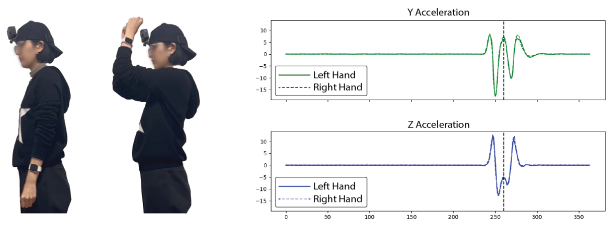


Figure 4. (a) Swinging arms for time synchronization. (b) IMU acceleration signals during sync. The vertical line indicates when the hand is up.

Watch OS offers unbiased acceleration values.

Camera Alignment to IMU Coordinates. For aligning camera coordinates to IMU coordinates, previous approaches [11, 23] leverage joint positions derived from IMU-based mocap modules for alignment. Different from such methods, our alignment method does not rely on IMU-derived body joint positions. At the start of recording, the user is instructed to put the camera in 4 positions defined on the x and y axis, denoted as I_x, I_y, of the global coordinate g defined in IMU calibration. (Fig. 1 (b)).

The corresponding camera positions in arbitrary coordinates from SLAM are denoted as $C_i, i \in \{1, ..., 4\}$. Alignment is done by finding the transformation matrix T_I^c which maps $\overrightarrow{C_1, C_2}$ to y-axis I_y and $\overrightarrow{C_3, C_4}$ to x-axis I_x. As the real-world data could be noisy and a closed-form solution may not exist, we formulate an optimization problem to find the optimal transformation matrix.

The scale of arbitrary coordinates of SLAM is set by measuring the distances in the calibration board in Fig. 1. The initial height h_0 is given by measuring the distance between the board and the floor the user stands on during the alignment procedure. The camera poses before and after alignment are in Fig. 2.

Head Poses From Camera Trajectory. The camera center $\mathbf{C}$ may not be necessarily the same as the head joint location, we compute the fixed transformation T_{head}^{cam} to transform the camera pose into the head pose. The translation component fo T_{head}^{cam} is computed by approximating the camera location in a surface point of SMPL mesh. As the camera may not be on the surface of the human head as in Fig. 4, an offset is added to the surface point. Similar to the T-pose calibration step in IMU sensor calibration (Supp. Sec. 3.2), the rotation component of T^{cam}_{head}, or $\mathbf{R}^{cam}_{head}$, is obtained by asking the user to stand still (as in T-pose) for 10 seconds.

Synchronization. The initial time synchronization between IMU sensors and the head-mounted camera is set by the timecodes generated from the sensors. (For the camera, the timecode is recorded when the shutter is activated) However, empirically we found there were minor errors in the timecode and therefore should be adjusted. For the adjustment, the user is asked to stand still and swing their arms up and down, as in Fig. 4 (a). When the hands move up to the highest position close to the camera (Fig. 4 (a)), the IMU sensor signals show a specific peak as in Fig. 4 (b). The time synchronization is done by aligning the two. To adjust between the camera and the IMU sensors, the frame where the hands are closest to the camera is selected. We empirically found out that swinging is more robust than clapping, as in some cases the force transmitted to the wrist sensor during clapping can cause sensor peaks to spike.

3.3. Capture Setup for Ablations

For quantitative evaluation of ablative baselines (Sec. 4.4) XSens MVN Link [5] was used to capture ground-truth data. As an IMU-based method, however, XSens also suffers from root drift issues [11]. We ignore these factors during the evaluation by considering errors in XY aligned space (in evaluating floor update algorithm) or capturing motion with relatively small root translations (in evaluating $\mathcal{F}_{opt}$).

For the ablation of floor plane update, the height changes from 0m to -4.21m. For ablations on visual-cue based motion optimization $\mathcal{F}_{opt}$ in egocentric cases, we capture 3 scenarios (making coffee, using coffee machine, taking snacks off the shelf; total 2289 frames) and the right hand was detected as the visual cue ϕ_E.

4. Implementation Details

4.1. Synthesizing IMU Sensor Data

For synthesizing IMU sensor data from motion capture datasets we follow the protocol in [13, 24, 25]. Rotations are obtained from joint rotations derived by solving forward kinematics. Accelerations are synthesized based on the following equation:

$$\mathbf{a}^j_t = \frac{\mathbf{p}^j_{t-n} + \mathbf{p}^j_{t+n} - 2\mathbf{p}^j_t}{(n\Delta t)^2} \quad j \in \{left, right\} \tag{3}$$

We set $n = 4$, which is reported [25] to synthesize sensor signals closest to the real sensors.

4.2. Floor Level Update

Implementation Details. The threshold λ for contact detection is set to 0.5. For projecting $\mathbf{p}^f_{t_m}$ to the pointcloud $\mathbf{W}$, set points $\{\mathbf{w}\}$ whose distance to $\mathbf{p}^f_{t_m}$ are less than 0.15m are searched. As the pointcloud $\mathbf{W}$ is not pre-scanned but is obtained from SLAM, the number of points may be sparse in floor regions. If the number N of points in $\{\mathbf{w}\}$ is below 10, the height of $\mathbf{p}^f_{t_m}$ is set as f_t. To prevent undesired updates, the floor level is not updated when (1) the floor level change is less than 0.1m, (2) both feet have been in contact from $t-5$ to $t-1$, and are still in contact for t to $t+5$, (3) the time interval between current and previous update is less than 25 frames.

4.3. Motion Estimation

Models and Training. For the transformer encoders in both submodules $\mathcal{F}^{end}$ and $\mathcal{F}^{body}$ the number of heads is set to 10, and the number of layers to 4. For $\mathcal{F}^{end}$, the input is projected into embeddings of dimensions 1280. For $\mathcal{F}^{body}$, mid-representations $\{\mathbf{x}^{mid}_\tau\}$ and $\{\mathbf{x}_\tau\}$ are projected into embeddings of dimension 640 each and are concatenated and fed into the Transformer encoder. The features generated by the Transformer encoder are converted to the final output of each module via a 2-layer MLP, which converts the features into 256 dimensions in the first layer and to the dimensions of the final output in the second layer. The length N of temporal sliding windows fed into the modules is set to 40. The two modules are trained end-to-end, with AdamW optimizer [15]. The learning rate is automatically set with DAdaptation [7]. We use Nvidia RTX 3090Ti GPU for training. PyTorch [19] and FairMotion library [10] were used for implementation.

Datasets. The datasets used for baseline comparison are stated in the main paper. For training the model used in real-world demonstrations in the supplementary video, we use a subset of AMASS (CMU [22], HDM05 [18], BMLMovi [8], KIT [17], HUMAN4D [6]) and a subset of Lafan1 [12] dataset for training.

4.4. Motion Optimization

Models and Training. The encoder and decoder of the autoencoder structure to build motion manifolds (Sec 3.4) which consist of 3 layers of 1D temporal-convolutions with a kernel width of 25 and stride 2. The channel dimension of each output feature is set to 256. The autoencoder is trained with the AdamW optimizer [15] and the learning rate is automatically set with DAdaptation [7].For optimizing the latent vector within the manifold, Adam optimizer [14] was used, and the learning rate was set to 0.0007 for single-person egocentric visual cues ϕ_E, 0.001 for multi-person visual cues ϕ_T. As in the motion estimation module, we use PyTorch [19] and FairMotion library [10] for implementation. Nvidia RTX 3090Ti GPU was used for training and optimization. The datasets used for training the autoencoder are identical to the datasets used in training $\mathcal{F}_{est}$ (Supp. Sec. 4.3).

Generating Visual Cues. MediaPipe library [16] was used for detecting 2D keypoints in single-person egocentric visual cues ϕ_E. The missing keypoints are tracked using the optical flow of the detected keypoints. State-of-the-art monocular 3D pose estimation method 4DHumans [9] was used for generating multi-person visual cues ϕ_T.

Reconstruction Loss. The encoder E and decoder E^{-1} are trained based on reconstruction loss $\mathcal{L}_{recon} = ||\mathbf{X} - E^{-1}(E(\mathbf{X}))||$, where:

$$\mathcal{L}_{recon} = \mathcal{L}_{contact} + \mathcal{L}_{root} + \mathcal{L}_{rot} + \mathcal{L}_{pos}. \tag{4}$$

$\mathcal{L}_{contact}$, $\mathcal{L}_{root}$, $\mathcal{L}_{rot}$, and $\mathcal{L}_{pos}$ are the L1 losses of foot contact labels, root translation and rotation, joint rotations in 6D representations [26], and global joint positions obtained by forward kinematics operation.

References

[1] Apple watch se. https://www.apple.com/apple-watch-se/. 2

[2] Core motion: Process accelerometer, gyroscope, pedometer, and environment-related events. https://developer.apple.com/documentation/coremotion. 2

[3] Open gopro python sdk. https://gopro.github.io/OpenGoPro/demos/python/sdk_wireless_camera_control. 2

[4] Sensorlog: Log and stream sensor data. https://sensorlog.berndthomas.net/. 2

[5] Xsens mvn link. https://www.movella.com/products/motion-capture/xsens-mvn-link. 4

[6] Anargyros Chatzitofis, Leonidas Saroglou, Prodromos Boutis, Petros Drakoulis, Nikolaos Zioulis, Shishir Subramanyam, Bart Kevelham, Caecilia Charbonnier, Pablo Cesar, Dimitrios Zarpalas, et al. Human4d: A human-centric multimodal dataset for motions and immersive media. *IEEE Access*, 8, 2020. 4

[7] Aaron Defazio and Konstantin Mishchenko. Learning-rate-free learning by d-adaptation. *arXiv preprint arXiv:2301.07733*, 2023. 4

[8] Saeed Ghorbani, Kimia Mahdaviani, Anne Thaler, Konrad Kording, Douglas James Cook, Gunnar Blohm, and Nikolaus F. Troje. MoVi: A Large Multipurpose Motion and Video Dataset, 2020. 4

[9] Shubham Goel, Georgios Pavlakos, Jathushan Rajasegaran, Angjoo Kanazawa, and Jitendra Malik. Humans in 4d: Reconstructing and tracking humans with transformers. In *ICCV*, 2023. 5

[10] Deepak Gopinath and Jungdam Won. fairmotion - tools to load, process and visualize motion capture data. Github, 2020. 4

[11] Vladimir Guzov, Aymen Mir, Torsten Sattler, and Gerard Pons-Moll. Human poseitioning system (hps): 3d human pose estimation and self-localization in large scenes from body-mounted sensors. In *CVPR*, 2021. 3, 4

[12] Félix G Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening. *ACM TOG*, 39(4), 2020. 4

[13] Yifeng Jiang, Yuting Ye, Deepak Gopinath, Jungdam Won, Alexander W Winkler, and C Karen Liu. Transformer inertial poser: Real-time human motion reconstruction from sparse imus with simultaneous terrain generation. In *SIGGRAPH Asia*, 2022. 1, 2, 3, 4

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 4

[16] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019. 5

[17] Christian Mandery, Ömer Terlemez, Martin Do, Nikolaus Vahrenkamp, and Tamim Asfour. Unifying representations and large-scale whole-body motion databases for studying human motion. *T-RO*, 32(4), 2016. 4

[18] Meinard Müller, Tido Röder, Michael Clausen, Bernhard Eberhardt, Björn Krüger, and Andreas Weber. Documentation mocap database hdm05. *Computer Graphics Technical Report CG-2007-2, Universität Bonn*, 2007. 4

[19] 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*. 2019. 4

[20] Zachary Teed and Jia Deng. DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras. In *NeurIPS*, 2021. 1

[21] Matthew Trumble, Andrew Gilbert, Charles Malleson, Adrian Hilton, and John Collomosse. Total capture: 3d human pose estimation fusing video and inertial sensors. In *BMVC*, 2017. 1

[22] Carnegie Mellon University. Cmu mocap dataset. 4

[23] Xinyu Yi, Yuxiao Zhou, Marc Habermann, Vladislav Golyanik, Shaohua Pan, Christian Theobalt, and Feng Xu. Egolocate: Real-time motion capture, localization, and mapping with sparse body-mounted sensors. *ACM TOG*, 42(4), 2023. 1, 2, 3

[24] Xinyu Yi, Yuxiao Zhou, Marc Habermann, Soshi Shimada, Vladislav Golyanik, Christian Theobalt, and Feng Xu. Physical inertial poser (pip): Physics-aware real-time human motion tracking from sparse inertial sensors. In *CVPR*, 2022. 1, 2, 4

[25] Xinyu Yi, Yuxiao Zhou, and Feng Xu. Transpose: Real-time 3d human translation and pose estimation with six inertial sensors. *ACM TOG*, 40(4), 2021. 2, 3, 4

[26] Yi Zhou, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. On the continuity of rotation representations in neural networks. In *CVPR*, 2019. 5