Supplementary Materials for "Real-Time Simulated Avatar from Head-Mounted Sensors"

Zhengyi Luo[1,2] Jinkun Cao[2] Rawal Khirodkar[1] Alexander Winkler[1] Jing Huang[1]
Kris Kitani[1,2,*] Weipeng Xu[1,*]
[1]Reality Labs Research, Meta; [2]Carnegie Mellon University
https://zhengyiluo.github.io/SimXR/

A. Introduction

In this supplement, we provide additional details about SimXR that are left out of the main paper due to space constraints. Specifically, in Sec. B, we describe the contents of our supplementary site and videos. In Sec. C, we discuss the implementation details of our proposed synthetic dataset (Sec C.1), our proposed method SimXR (Sec C.2), pretrained imitators that act as teachers (Sec C.3), KinPoly-v (Sec C.4), and UnrealEgo (Sec C.5). Finally, in Sec. D, we provide some additional ablations (such as using recurrent networks) and analysis of failure cases. Since motion is best seen in videos, we strongly encourage our readers to view the provided videos for a qualitative analysis of our method. **All data and models will be released.**

B. Supplementary Site & Videos

In the supplement site, we provide an extensive qualitative evaluation of our method. We visualized all three subjects and **full** sequences of our real-world Quest 2 data capture, as well as results from the synthetic dataset. From the videos, we can see that our end-to-end method can follow the headset wearer's body motion closely in a physically plausible fashion. We also show results on AR/Aria glasses and compare with SOTA vision-based and physics-based methods. Last but not least, we visualize failure cases of our method.

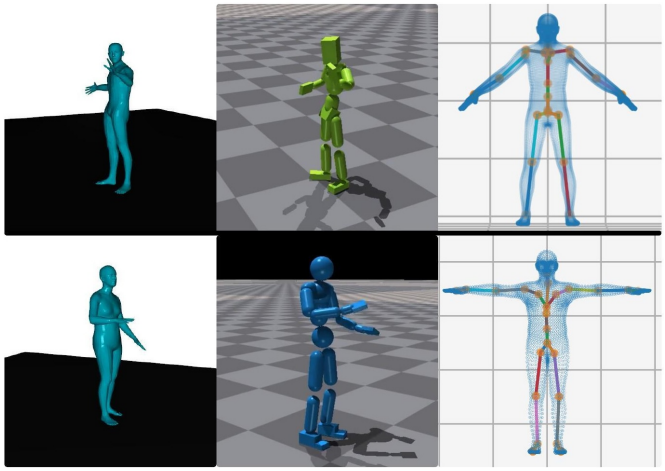

Figure 1. The two human models we use, their rendered mesh, simulated humanoid, and kinemaitc structure. *(Top)*: our in-house humanoid with 24 DOF. *(Bottom)*: SMPL humanoid with 23 DOF.

C. Implementation Details

C.1. Synthetic Data Generation

Humanoid Kinematic Structure. To create the synthetic egocentric data, we use an internal human mesh model similar to SMPL [3]. Given kinematic body rotations and scale parameters, we can create its corresponding mesh as shown in Fig. 1. We use a process similar to that of the SMPL humanoid to create an IsaacGym compatible humanoid for the in-house human mesh model.

MoCap Dataset. To generate synthetic data, we use a large-scale internal MoCap dataset consisting of 130 subjects and $>$ 1300 capture sessions. The motion capture dataset contains a large number of daily activities (walking, running, gesturing, yoga, dancing, balancing, sitting, interaction with objects *etc*.). We remove sequences that contain

*Equal advising.

Figure 2. Self-occlusion visualization; blue dots are keypoints.

human-object interactions that are not possible to mimic without simulating the objects (such as sitting on chairs). Since each capture session contains a long sequence of motion, we further divide the sessions into sequences that contain around ∼450 frames of motion, resulting in a total of 5169 sequences for training and testing.

Rendering Pipeline. As mentioned in the main paper, rendering is done using the exact placement, intrinsic, and distortion of the cameras as the Quest 2 headset. The Unity [1] game engine is used for rendering. In Fig. 3, we show examples of raw RGB images rendered using our pipeline. In each frame, we randomize the clothing, lighting, and background of the subjects as domain randomization. The background is rendered using a random image projected onto a skybox. We render each frame of motion from the MoCap dataset in 30 FPS. Each RGB image is rendered in a $640 \times 480 \times 3$ resolution, and we convert the images to monochrome and shrink them to $160 \times 120 \times 1$ for training and testing `SimXR`.

Synthetic-Test, $E_{\text{pa-mpjpe}} \downarrow$						
	Head	L_Shoulder	L_Elbow	L_Wrist	R_Shoulder	R_Elbow
In-frame %	0.0%	4.0%	61.3 %	92.4 %	18.1 %	75.8%
In-frame / Out-frame	- / 30.2	28.0 / **25.0**	**30.7** / 33.2	**42.7** / 86.3	25.6 / **24.3**	**30.4** / 34.3
	R_Wrist	L_Knee	L_Ankle	R_Knee	R_Ankle	
In-frame %	96.5 %	99.0 %	98.4 %	99.4%	98.9%	
In-frame / Out-frame	**41.1** / 94.2	43.9 / **39.9**	**60.4** / 74.4	**43.7** / 45.5	**60.3** / 72.7	

Table 1. Error analysis based on joint in-frame status. "In-frame" is not equivalent to "visible" due to self-occlusion.

Visibility Analysis. Here we conduct a visibility analysis on our generated synthetic data. As accounting for self-occlusion involves reprocessing the synthetic dataset and conducting ray-marching for each joint on the clothed mesh to ascertain visibility, we opt to use the "in-frame" (whether the joint is in one of the camera frames) statistics to approximate visibility. This measure is a reliable visibility indicator for upper body joints, but less so for the lower body, where self-occlusion and extreme viewpoints are more prevalent (see Figure 2). From Table 1 we can see that the shoulder and elbow joints are frequently outside the frame, while the wrist and leg joints often remain within the frame. For the shoulder, elbow, and knee joints, their in-frame and out-frame results are similar, since they are closer to the torso. For the body extremities (wrists and ankles), there is a clear gap, as the largest errors occur out of the frame.

C.2. Details about `SimXR`

Body Shape Used for Evaluation. We conduct all our training and evaluation using a fixed body shape for both of our humanoids (SMPL and in-house). In other words, we use the mean body shape for SMPL and a fixed body shape for our internal humanoid and do not vary bone lengths between different motion sequences or subjects. Since our framework does not involve any intermediate representations such as 3D keypoints or poses, `SimXR` is scale invariant. When conducting real-world evaluations, we simply adjust the height of the headset pose to match the standing head positions of the mean body shape. This is done in a calibration phase in which the subject is standing still. This process is effective as `SimXR` can estimate the pose for the three subjects who have different heights. Notice that our imitator can be trained to handle different body shapes, but we opt out of this option as estimating body shape from the distorted egocentric views is still an unsolved problem.

	Batch Size	Learning Rate	# of samples	image size	Image-latent
`SimXR`	1024	5×10^{-4}	$\sim 10^8$	160×120	512
	Batch Size	Learning Rate	# of samples		
PHC	3072	2×10^{-5}	$\sim 10^{10}$		

Table 2. Hyperparameters for `SimXR` and PHC. Due to the increase in input size, `SimXR` is trained with significantly less samples than PHC and requires distillation.

Training Process. The training process for `SimXR` is similar to training a motion imitator, with the distinction being that we provide images and headset pose as input instead of full-body reference pose. To better learn harder motion sequences, we use the same hard-negative mining process proposed in PHC [7] and PULSE [6]. Concretely, during training, given the full motion and image dataset $\hat{\boldsymbol{Q}}$, we evaluate the current policy on the full dataset and pick the sequences that the policy fails to form $\hat{\boldsymbol{Q}}_{\text{hard}}$. We keep updating $\hat{\boldsymbol{Q}}_{\text{hard}}$ at intervals until the success rate no longer increases. The hyperparameters for training `SimXR` can be found in Table 2.

C.3. Details about Pretrained Imitators

For the SMPL humanoid (AR / Aria glass experiments), we use an off-the-shelf motion imitator, PHC [7], trained on the AMASS dataset. We do not make any additional modifications to PHC, since the motion in the ADT dataset is relatively simple. For the in-house humanoid and VR / Quest experiments, we train an imitator using the same training procedure and hyperparameters provided in the PHC implementation. We train the imitator using the training sequences from the internal MoCap dataset, achieving the imitation performance shown in Table 3. We can see that the imitator has a high success rate and a low joint error on the training data, which means that it is suitable to be used as a teacher for downstream tasks.

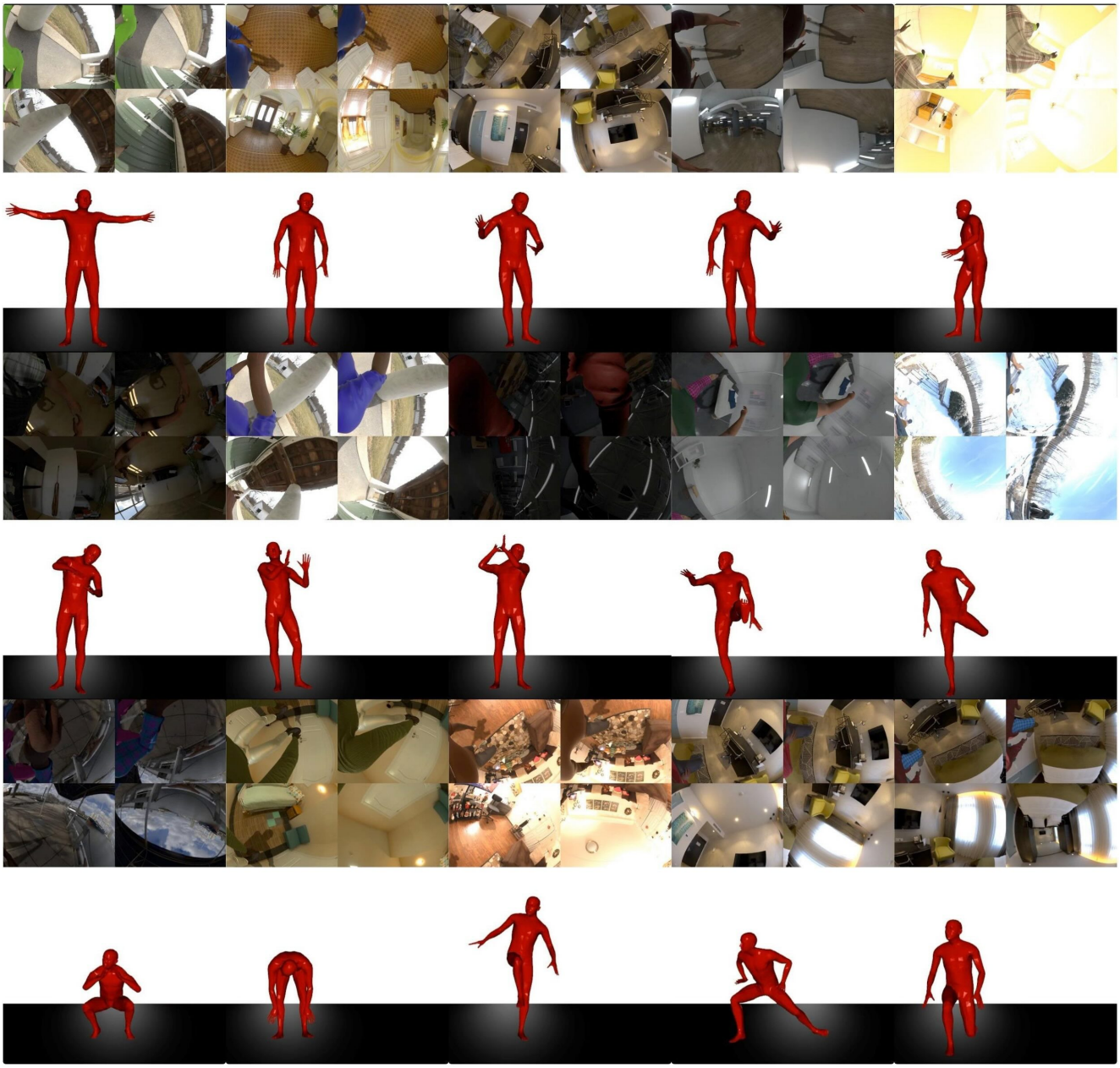

Figure 3. Sample synthetic data with various poses. Here we include the original rendered RGB images for demonstration purposes. We randomize the actor's clothes, background, lighting at every frame.

	Synthetic-Train					
Method	Succ ↑	$E_{\text{g-mpjpe}}$ ↓	E_{mpjpe} ↓	$E_{\text{pa-mpjpe}}$ ↓	E_{acc} ↓	E_{vel} ↓
PHC	99.8%	25.6	20.4	15.5	2.4	3.5

Table 3. Motion imitation result by the pretrained imitator on the in-house MoCap dataset.

C.4. Details about KinPoly-v

We adapt KinPoly [4] to also consume images as input for egocentric pose estimation. In KinPoly, a policy is learned to produce kinematic full-body poses $\tilde{\boldsymbol{q}}_{t+1}$ based on the pose of the headset, which is then fed to an external force based motion imitator (UHC) for physics-based imitation. Comparing KinPoly to previous methods that use both an imitator and a pose estimator (*e.g.* SimPOE [10]), the main difference is whether the pose estimator is aware of the simulated humanoid state. In prior art, the pose estimator is not conditioned on the simulation state and operates independently from the physics simulation. This creates an open-loop system where the pose estimator can estimate a pose that drifts far away from the simulation state, leading the

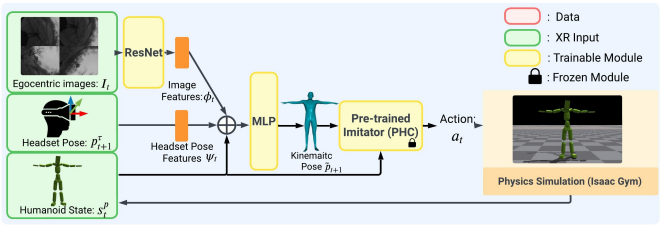

Figure 4. KinPoly-v's network architecture. Different from distilling from a pretrained imitator, KinPoly-v outputs kinematic pose to the pretrained imitator for physcis-based motion imtiation.

imitator to fall. KinPoly aims to create a closed-loop system where the pose estimator also takes the simulation state into consideration. This methodology is also used in EmbodiedPose [5]. However, the main problem with KinPoly is that, while it is relatively easy to output the correct reference kinematic pose $\tilde{\boldsymbol{q}}_{t+1}$ for the current timestep, it is difficult to compute the correct velocities $\tilde{\dot{\boldsymbol{q}}}_{t+1}$. Due to the exceptional capabilities of the non-physical forces, UHC does not require reference velocities as input. Concretely, UHC's goal state is defined as $\boldsymbol{s}_t^{\text{g-mimic}} \triangleq (\hat{\boldsymbol{\theta}}_{t+1} \ominus \boldsymbol{\theta}_t, \hat{\boldsymbol{p}}_{t+1} - \boldsymbol{p}_t)$, which does not contain any velocity information. As a result, KinPoly's kinematic policy only needs to predict body pose, but not velocity, which simplifies the learning problem. However, since PHC does not use any external forces [9], it requires accurate reference velocities as input.

To remove the dependency on external forces (change from UHC to PHC), we experimented with two forms of velocity prediction. The first approach is to compute the velocity as a finite difference between consecutive frames of the predicted reference poses: $\tilde{\dot{\boldsymbol{q}}}_{t+1} = \tilde{\boldsymbol{q}}_{t+1} - \tilde{\boldsymbol{q}}_t$. This formulation is problematic as large jumps in predicted poses can result in large velocities, which in turn lead to the imitator falling. Another approach is to directly predict the velocity as an output, which turns out to be more stable. We use this version as our implementation for KinPoly-v. However, this approach still suffers from inaccurate velocity prediction, as can be seen in the supplement videos: when the motion becomes faster and more dynamic (such as sports movement or jogging), it becomes difficult to predict the correct velocities for the motion imitator. This can also be a result of the image input, which can be noisy and detrimental to the network learning a good velocity prediction.

Network Architecture. KinPoly-v shares the same input and network architecture as SimXR, with the only distinction being the output: KinPoly-v outputs the kinematic pose $\tilde{\boldsymbol{q}}_{t+1}$ rather than the PD targets $\boldsymbol{a}_t$ for the joints. KinPoly-v's architecture can be found in Fig.4

C.5. Details about UnrealEgo

We use the official UnrealEgo implementation, and pick the ResNet18 version with imagenet initialization for a fair comparison with SimXR. To be compatible with monochrome images, we replace the CNN layer with a single-channel convolutional layer and keep the Siamese network structure. We follow the official implementation and first train the 2D heatmap estimation network. Then, using the frozen heatmap estimation network, we train a 3D pose estimator based on the 2D heatmap input. We train the networks for three days to convergence, using a similar compute budget as training SimXR.

	Synthetic-Test					
Method	Succ ↑	$E_{\text{g-mpjpe}}$ ↓	E_{mpjpe} ↓	$E_{\text{pa-mpjpe}}$	E_{acc} ↓	E_{vel} ↓
Ours (GRU)	91.6%	78.2	73.7	52.8	8.8	10.4
Ours	**96.2%**	**69.0**	**65.2**	**43.2**	**6.8**	**8.7**

Table 4. Additional ablation on using recurrent architecture

D. Additional Ablations and Failure Cases

Recurrent Networks. Currently, SimXR is a per-frame model without using any temporal model architecture. SimXR does rely on temporal information in the form of a simulation state and estimates temporally coherent motion by jointly considering humanoid state and input images. Based on the intuition that incorporating recurrent networks is essential to help robots complete tasks [11], we also tried recurrent architecture in our early experiments. We tested a simple GRU-based [2] architecture with 512 hidden units, and forms a lightweight Conv-LSTM [8]. Table 4 shows that the use of a recurrent network does not offer an immediate performance increase. We hypothesize that, for a pose estimation task with dense per-frame input, a recurrent network may not be necessary to ensure good performance. Further investigation is needed to better leverage the temporal coherence in videos.

Additional Failure Cases. In our supplementary videos, we visualize the common failure cases of SimXR. Being one of the first methods to drive simulated avatars from images and headset pose input from XR headsets, SimXR shows the feasibility of training such a network, but it is still far from perfect.

- We can observe that the humanoid stumbles and drags its feet to stay balanced when moving around, which is caused by ambiguity in movement signals. Since our humanoid has no information about the future movements of the camera wearer, it adopts the foot-dragging behavior to be cautious and stay balanced.
- Accurate foot movement can still be challenging: while SimXR can estimate kicking and raising feet, it can also miss raised feet due to the challenging viewing angle. The foot can be barely visible even when raised, and the color of the garment can create additional ambiguities.
- We can observe that when the hands are held perfectly

still, the humanoid can have micro movements due to inaccuracy in inferring the body poses. Tackling this issue is challenging, as the movement of the headset can cause the hands to move in the camera space but not in the global space, and differentiating between the two requires further investigation.

- The humanoid can also have erroneous hands movement from time to time, erecting the hand quickly and putting them down due to image noise and occlusion.
- Another source of inaccuracy is fast and sporty movement, where the humanoid can lag behind in performing the actions or fall down.

In the future, our aim is to incorporate a larger MoCap and synthetic datasets to improve the robustness of the controller. Introducing auxiliary pose estimation losses during training could also improve `SimXR`.

Acknowledgement. We thank Zihui Lin for her help in making the plots in this paper. Zhengyi Luo is supported by the Meta AI Mentorship (AIM) program.

References

[1] Beta Program. Unity real-time development platform. https://unity.com/. Accessed: 2023-11-18. 2

[2] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014. 4

[3] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM Trans. Graph.*, 34, 2015. 1

[4] Zhengyi Luo, Ryo Hachiuma, Ye Yuan, and Kris Kitani. Dynamics-regulated kinematic policy for egocentric pose estimation. *NeurIPS*, 34:25019–25032, 2021. 3

[5] Zhengyi Luo, Shun Iwase, Ye Yuan, and Kris Kitani. Embodied scene-aware human pose estimation. *NeurIPS*, 2022. 4

[6] Zhengyi Luo, Jinkun Cao, Josh Merel, Alexander Winkler, Jing Huang, Kris Kitani, and Weipeng Xu. Universal humanoid motion representations for physics-based control. *arXiv preprint arXiv:2310.04582*, 2023. 2

[7] Zhengyi Luo, Jinkun Cao, Alexander W. Winkler, Kris Kitani, and Weipeng Xu. Perpetual humanoid control for real-time simulated avatars. In *International Conference on Computer Vision (ICCV)*, 2023. 2

[8] Tara N. Sainath, Oriol Vinyals, Andrew W. Senior, and Hasim Sak. Convolutional, long short-term memory, fully connected deep neural networks. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584, 2015. 4

[9] Ye Yuan and Kris Kitani. Residual force control for agile human behavior imitation and extended motion synthesis. *arXiv preprint arXiv:2006.07364*, 2020. 4

[10] Ye Yuan, Shih-En Wei, Tomas Simon, Kris Kitani, and Jason Saragih. Simpoe: Simulated character control for 3d human pose estimation. *CVPR*, 2021. 3

[11] Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023. 4