Supplmentary Material for “Perpetual Humanoid Control for Real-time Simulated Avatars”

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

A. Introduction

In this document, we include additional details and results that are not included in the paper due to the page limit. In Sec.B, we include additional details for training, avatar use cases, and progressive neural networks (PNN) [8]. In Sec.C, we include additional ablation results. Finally, in Sec.D, we provide an extended discussion of limitations, failure cases, and future work.

Extensive qualitative results are provided on the project page. We highly encourage our readers to view them to better understand the capabilities of our method. Specifically, we show our method’s ability to imitate high-quality MoCap data (both train and test) and noisy motion estimated from video. We also demonstrate real-time video-based (single- and multi-person) and language-based avatar (single- and multiple-clips) use cases. Lastly, we showcase our fail-state recovery ability.

B. Implementation Details

B.1. Training Details

Humanoid Construction. Our humanoid can be constructed from any kinematic structure, and we use the SMPL humanoid structure as it has native support for different body shapes and is widely adopted in the pose estimation literature. Fig.1 shows our humanoid constructed based on randomly selected gender and body shape from the AMASS dataset. The simulation result can then be exported and rendered as the SMPL mesh. We showcase two types of constructed humanoid: capsule-based and mesh-based. The capsule-based humanoid is constructed by treating body parts as simple geometric shapes (spheres, boxes, and capsules). The mesh-based humanoid is constructed following a procedure similar to SimPoE[11], where each body part is created by finding the convex hull of all vertices assigned to each bone. The capsule humanoid is easier to simulate and design, whereas the mesh humanoid provides a better approximation of the body shape to simulate more complex human-object interactions. We find that mesh-based and capsule-based humanoids do not have significant performance differences (see Sec.C) and conduct all experiments using the capsule-based humanoid. For a fair comparison with the baselines, we use the mean body shape of the SMPL with neutral gender for all evaluations and show qualitative results for shape variation. For both types of humanoids, we scale the density of geometric shapes so that the body has the correct weight (on average 70 kg). All inter-joint collisions are enabled for all joint pairs except for between parent and child joints. Collision between humanoids can be enabled and disabled at will (for multi-person use cases).

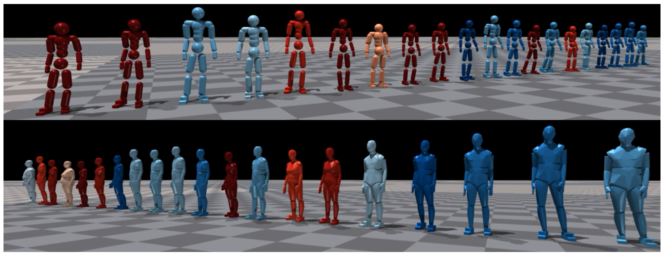

Figure 1: Our framework can support body shape and gender variations. Here we showcase humanoids of different gender and body proportion holding a standing pose. We construct two kinda of humanoids: capsule-based (top) and mesh-based (bottom). Red: female, Blue: male. Color gradient indicates weight.

Training Process. During training, we randomly sample

Table 1: Hyperparameters for PHC. σ: fixed variance for policy. γ: discount factor. ϵ: clip range for PPO

	Batch Size	Learning Rate	σ	γ	ϵ
Value	1536	5×2^{-5}	0.05	0.99	0.2
	w_{jp}	w_{jr}	w_{jv}	$w_{\text{j}\omega}$	
Value	0.5	0.3	0.1	0.1	

motion from the current training set $\hat{\boldsymbol{Q}}^{(k)}$ and normalize it with respect to the simulated body shape by performing forward kinematics using $\hat{\boldsymbol{\theta}}_{1:T}$. Similar to UHC [5], we adjust the height of the root translation $\hat{\boldsymbol{p}}^0_t$ to make sure that each of the humanoid's feet touches the ground at the beginning of the episode. We use parallelly simulate 1536 humanoids for training all of our primitives and composers. Training takes around 7 days to collect approximately 10 billion samples. When training with different body shapes, we randomly sample valid human body shapes from the AMASS dataset and construct humanoids from them. Hyperparamters used during training can be found in Table.1

Data Preparation. We follow similar procedure to UHC [5] to filter out AMASS sequences containing human object interactions. We remove all sequences that sits on chairs, move on treadmills, leans on tables, steps on stairs, floating in the air *etc*., resulting in 11313 high-quality motion sequences for training and 140 sequences for testing. We use a heuristic-based filtering process based on *i.e*. identifying the body joint configurations corresponding to the sitting motion or counting number of consecutive airborne frames.

Runtime. Once trained, our PHC can run in real time ($\sim$ 32FPS) together with simulation and rendering, and around ($\sim$ 50FPS) when run without rendering. Table.2 shows the runtime of our method with respect to the number of primitives, architecture, and humanoid type used.

Model Size. The final model size (with four primitives) is 28.8 MB, comparable to the model size of UHC (30.4 MB).

B.2. Real-time Use Cases

Real-time Physics-based Virtual Avatars from Video. To achieve real-time physics-based avatars driven by video, we first use Yolov8[3] for person detection. For pose estimation, we use MeTRAbS [9] and HybrIK [4] to provide 3D keypoints $\tilde{\boldsymbol{p}}_t$ and rotation $\tilde{\boldsymbol{\theta}}_t$. MeTRAbs is a 3D keypoint estimator that computes 3D joint positions $\tilde{\boldsymbol{p}}_t$ in the absolute global space (rather than in the relative root space). HybrIK is a recent method for human mesh recovery and computes joint angles $\tilde{\boldsymbol{\theta}}_t$ and root position $\tilde{\boldsymbol{p}}^0_t$ for the SMPL human body. One can recover the 3D keypoints $\tilde{\boldsymbol{p}}_t$ from joint angles $\tilde{\boldsymbol{\theta}}_t$ and root position $\tilde{\boldsymbol{p}}^0_t$ using forward kinematics. Both of these methods are causal, do not use any temporal information, and can run in real-time ($\sim$ 30FPS). Estimating 3D keypoint location from image pixels is an easier task than regressing joint angles, as 3D keypoints can be better associated with features learned from pixels. Thus, both HybrIK and MeTRAbs estimate 3D keypoints $\tilde{\boldsymbol{p}}_t$, with HybrIK containing an additional step of performing learned inverse kinematics to recover joint angles $\tilde{\boldsymbol{\theta}}_t$. We show results using both of these off-the-shelf pose estimation methods, using MeTRAbs with our keypoint-based controller and HybrIK with our rotation-based controller. Empirically, we find that MeTRAbs estimates more stable and accurate 3D keypoints, potentially due to its keypoint-only formulation. We also present a real-time **multi-person** physics-based human-to-human interaction use case, where we drive multiple avatars and enable inter-humanoid collision. To support multi-person pose estimation, we use OCSort [1] to track individual tracklets and associate poses with each person. Notice that real-time use cases pose additional challenges than offline processing: detection, pose/keypoint estimation, and simulation all need to run at real-time at around 30 FPS, and small fluctuations in framerate could lead to unstable imitation and simulation. To smooth out noisy depth estimates, we use a Gaussian filter to smooth out estimates from t-120 to t, and use the "mirror" setting for padding at boundary.

Virtual Avatars from Language. For language-based motion generation, we adopt MDM [10] as our text-to-motion model. We use the official implementation, which generates 3D keypoints $\tilde{\boldsymbol{p}}_t$ by default and connects it to our keypoint-based imitator. MDM generates fixed-length motion clips, so additional blending is needed to combine multiple clips of generated motion. However, since PHC can naturally go to far-away reference motion and handles disjoint between motion clips, we can naively chain together multiple clips of motion generated by MDM and create coherent and physically valid motion from multiple text prompts. This enables us to create a simulated avatar that can be driven by a continuous stream of text prompts.

B.3. Progressive Neural Network (PNN) Details

A PNN [8] starts with a single primitive network $\boldsymbol{\mathcal{P}}^{(1)}$ trained on the full dataset $\hat{\boldsymbol{Q}}$. Once $\boldsymbol{\mathcal{P}}^{(1)}$ is trained to convergence on the entire motion dataset $\hat{\boldsymbol{Q}}$ using the imitation task, we create a subset of hard motions by evaluating $\boldsymbol{\mathcal{P}}^{(1)}$ on $\hat{\boldsymbol{Q}}$. Sequences that $\boldsymbol{\mathcal{P}}^{(1)}$ fails forms $\hat{\boldsymbol{Q}}^{(1)}_{\text{hard}}$. We then freeze the parameters of $\boldsymbol{\mathcal{P}}^{(1)}$ and create a new primitive $\boldsymbol{\mathcal{P}}^{(2)}$ (randomly initialized) along with lateral connections that connect each layer of $\boldsymbol{\mathcal{P}}^{(1)}$ to $\boldsymbol{\mathcal{P}}^{(2)}$. Given the layer weight $\boldsymbol{W}_i^{(k)}$, activation function f, and the learnable lateral connection weights $\boldsymbol{U}_i^{(j:k)}$, we have the hidden activation $\boldsymbol{h}_i^{(k)}$ of the i^{th} layer of k^{th} primitive as:

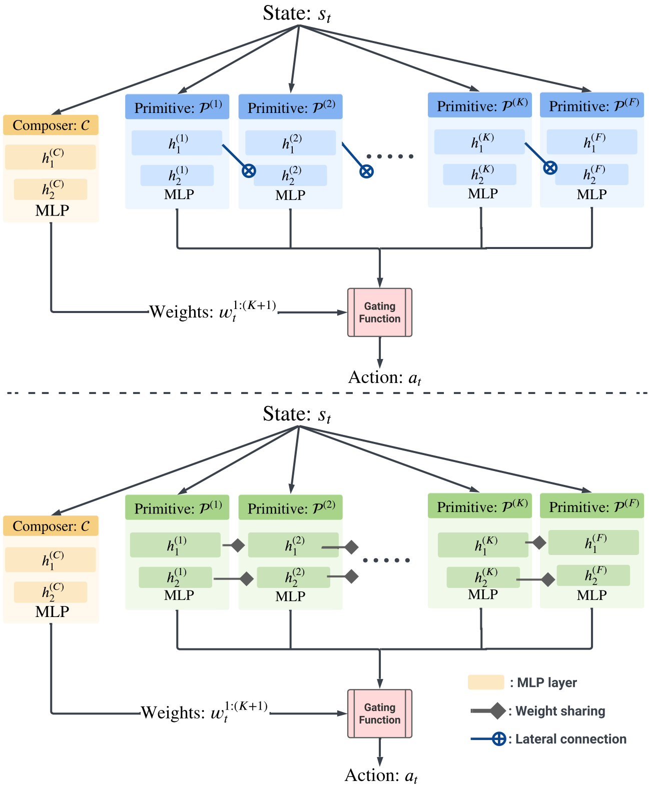

Figure 2: Progressive neural network architecture. Top: PNN with lateral connection. Bottom: PNN with weight sharing. $h_i^{(j)}$ indicates hidden activation of j^{th} primitive's i^{th} layer.

$$\boldsymbol{h}_i^{(k)} = f\left(\boldsymbol{W}_i^{(k)}\boldsymbol{h}_{i-1}^{(k)} + \sum_{j<k}\boldsymbol{U}_i^{(j:k)}\boldsymbol{h}_{i-1}^{(j)}\right). \tag{1}$$

Fig.2 visualizes the PNN with the lateral connection architecture. Essentially, except for the first layer, each subsequent layer receives the activation of the previous layer processed by the learnable connection matrices $\boldsymbol{U}_i^{(j:k)}$. We do not use any adapter layer as in the original paper. As an alternative to lateral connection, we explore weight sharing and warm-starts the primitive with the weights from the previous one (as opposed to randomly initialized). We find both methods equally effective (see Sec.C) when trained with the same hard-negative mining procedure, as each newly learned primitive adds new sequences that PHC can imitate. The weight sharing strategy significantly decreases training time as the policy starts learning harder sequences with basic motor skills. We use weight sharing in all our main experiments.

C. Supplementary Results

C.1. Categorizing the Forgetting Problem

As mentioned in the main paper, one of the main issues in learning to mimic a large motion dataset is the forgetting problem. The policy will learn new sequences while forgetting the ones already learned. In Fig.3, we visualize the sequences that the policy fails to imitate during training. Starting from the 12.5k epoch, each evaluation shows that some sequences are learned, but the policy will fail on some already learned sequences. The staircase pattern indicates that when learning sequences failed previously, the policy forgets already learned sequences. Numerically, each evaluation has around 30% overlap of failed sequences (right end side). The 30% overlap contains the backflips, cartwheeling, and acrobatics; motions that the policy consistently fails to learn when trained together with other sequences. We hypothesize that these remaining sequences (around 40) may require additional sequence-level information for the policy to learn properly together with other sequences.

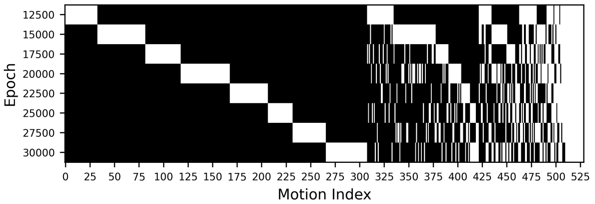

Figure 3: Here we plot the motion indexes that the policy fails on over training time; we only plot the 529 sequences that the policy has failed on over these training epoches. A white pixel denotes that sequence is can be successfully imitated at the given epoch, and a black pixel denotes an unsuccessful imitation. We can see that while there are 30 sequences that the policy consistently fails on, the remaining can be learned and then forgotten as training progresses. The staircase pattern indicates that the policy fails on new sequences each time it learns new ones.

Fail-state recovery Learning the fail-state recovery task can also lead to forgetting previously learned imitation skills. To verify this, we evaluate $\boldsymbol{\mathcal{P}}^{(F)}$ on the H36M-Test-Video dataset, which leads to a performance of Succ: 42.5%, $E_{\text{g-mpjpe}}$: 87.3, and E_{mpjpe}: 55.9, which is much lower than the single primitive $\boldsymbol{\mathcal{P}}^{(1)}$ performance of Succ: 59.4%, $E_{\text{g-mpjpe}}$: 60.2, and E_{mpjpe}: 34.4. Thus, learning the fail-state recovery task may lead to severe forgetting of the imitation task, motivating our PMCP framework to learn separate primitives for imitation and fail-state recovery.

C.2. Additional Ablations

In this section, we provide additional ablations of the components of our framework. Specifically, we study the effect of MOE vs. MCP, lateral connection vs. weight sharing, and the number of primitives used. We also report the inference speed (counting network inference and simulation time). All experiments are carried out with the rotation-based imitator and incorporate the fail state recovery primitive $\boldsymbol{\mathcal{P}}^{(F)}$ as the last primitive.

PNN Lateral Connection vs. Weight Sharing. As can be seen in Table 2, comparing Row 1 (R1) and R7 , we can see that PNN with lateral connection and weight sharing produce similar performance, both in terms of motion imitation and inference speed. This shows that *in our setup*, the weight sharing scheme is an effective alternative to lateral connections. This can be explained by the fact that in our case, each "task" on which the primitives are trained is similar and does not require lateral connection to choose whether to utilize prior experiences or not.

MOE vs. MCP. The difference between the top-1 mixture of experts (MOE) and multiplicative control (MCP) is discussed in

Table 2: Supplmentary ablation on components of our pipeline, performed using noisy pose estimate from HybrIK + Metrabs (root) on the H36M-Test-Video* data. MOE: top-1 mixture of experts. MCP: multiplicative control policy. PNN: progressive neural networks. Type: between Cap (capsule) and mesh-based humanoids. All models are trained with the same procedure.

						H36M-Test-Video*			
PNN-Lateral	PNN-Weight	MOE	MCP	Type	# Prim	Succ ↑	$E_{\text{g-mpjpe}}$ ↓	E_{mpjpe} ↓	FPS
✓	✗	✗	✗	Cap	4	87.5%	55.7	36.2	32
✗	✓	✓	✗	Cap	4	87.5%	56.3	34.3	33
✗	✓	✗	✓	Mesh	4	86.9%	62.6	39.5	30
✗	✗	✗	✗	Cap	1	59.4%	60.2	37.2	32
✗	✓	✗	✓	Cap	2	65.6%	58.7	37.3	32
✗	✓	✗	✓	Cap	3	80.9%	56.8	36.1	32
✗	✓	✗	✓	Cap	4	**88.7%**	**55.4**	**34.7**	32
✗	✓	✗	✓	Cap	5	87.5%	57.7	36.0	32

detail in the MCP paper [6]: top-1 MOE only activates one expert at a time, while MCP can activate all primitives at the same time. Comparing R2 and R7, as expected, we can see that top-1 MOE is slightly inferior to MCP. Since all of our primitives are pretrained and frozen, theoretically a perfect composer should be able to choose the best primitive based on input for both MCP and MOE. MCP, compared to MOE, can activate all primitives at once and search a large action space where multiple primitives can be combined. Thus, MCP provides better performance, while MOE is not far behind. This is also observed by CoMic[2], where they observe similar performance between mixture and product distributions when used to combine subnetworks. Note that top-inf MOE is similar to MCP where all primitives can be activated.

Capsule vs. Mesh Humanoid. Comparing R3 and R7, we can see that mesh-based humanoid yield similar performance to capsule-based ones. It does slow down simulation by a small amount (30 FPS vs. 32 FPS), as simulating mesh is more compute-intensive than simulating simple geometries like capsules.

Number of primitives. Comparing R4, R5, R6, R7, and R8, we can see that the performance increases as the number of primitives increases. Since the last primitive $\boldsymbol{\mathcal{P}}^{(F)}$ is for fail-state recovery and does not provide motion imitation improvement, R5 is similar to the performance of models trained without PMCP (R4). As the number of primitives grows from 2 to 3, we can see that the model performance grows quickly, showing that MCP is effective in combining pretrained primitives to achieve motion imitation. Since we are using relatively small networks, the inference speed does not change significantly with the number of primitives used. We notice that as the number of primitives grows, $\hat{\boldsymbol{Q}}^{(k)}$ becomes more and more challenging. For instance, $\hat{\boldsymbol{Q}}^{(4)}$ contains mainly highly dynamic motions such as high-jumping, back flipping, and cartwheeling, which are increasingly difficult to learn together. We show that (see supplementary webpage) we can overfit these sequences by training on them only, yet it is significantly more challenging to learn them together. Motions that are highly dynamic require very specific steps to perform (such as moving while airborne to prepare for landing). Thus, the experiences collected when learning these sequences together may contradict each other: for example, a high jump may require a high speed running up, while a cartwheel may require a different setup of foot-movement. A per-frame policy that does not have sequence-level information may find it difficult to learn these sequences together. Thus, sequence-level or information about the future may be required to learn these high dynamic motions together. In general, we find that using 4 primitives is most effective in terms of training time and performance, so for our main evaluation and visualizations, we use **4-primitive models**.

D. Extended Limitation and Discussions

Limitation and Failure Cases. As discussed in the main paper, PHC has yet to achieve 100% success rate on the AMASS training set. With a 98.9% success rate, PHC can imitate *most* of our daily motion without losing balance, but can still struggle to perform more dynamic motions, such as backflipping. For our real-time avatar use cases, we can see a noticeable degradation in performance from the offline counterparts. This is due to the following:

- Discontinuity and noise in reference motion. The inherent ambiguity in monocular depth estimation can result in noisy and jittery 3D keypoints, particularly in the depth dimension. These small errors, though sometimes imperceptible to the human eye, may provide PHC with incorrect movement signals, leaving insufficient time for appropriate reactions. Velocity estimation is also especially challenging in real-time use cases, and PHC relies on stable velocity estimation to infer movement cues.
- Mismatched framerate. Since our PHC assumes 30 FPS motion input, it is essential for pose estimates from video to match for a more stable imitation. However, few pose estimators are designed to perform real-time pose estimation ($\geq$ 30 FPS), and the estimation framerate can fluctuate due to external reasons, such as the load balance on computers.
- For multi-person use case, tracking and identity switch can still happen, leading to a jarring experience where the humanoids need to switch places.

A deeper integration between the pose estimator and our controller is needed to further improve our real-time use cases. As we do not explicitly account for camera pose, we assume that the webcam is level with the ground and does not contain any pitch or roll. Camera height is manually adjusted at the beginning of the session. The pose of the camera can be taken into account in the pose estimation stage. Another area of improvement is naturalness during fail-state recovery. While our controller can recover from fail-state in a human-like fashion and walks back to resume imitation, the speed and naturalness could be further improved. Walking gait, speed, and tempo during fail-state recovery exhibits noticeable artifacts, such as asymmetric motion, a known artifact in AMP [7]. During the transition between fail-state recovery and motion imitation, the humanoid can suddenly jolt and snap into motion imitation. Further investigation (*e.g.* better reward than the point-goal formulation, additional observation about trajectory) is needed.

Discussion and Future Work. We propose the perpetual humanoid controller, a humanoid motion imitator capable of imitating large corpus of motion with high fidelity. Paired with its ability to recover from fail-state and go back to motion imitation, PHC is ideal for simulated avatar use cases where we no longer require reset during unexpected events. We pair PHC with a real-time pose estimator to show that it can be used in a video-based avatar use case, where the simulated avatar imitates motion performed by the actors perpetually without requiring reset. This can empower future virtual telepresence and remote work, where we can enable physically realistic human-to-human interactions. We also connect PHC to a language-based motion generator to demonstrate its ability to mimic generated motion from text. PHC can imitate multiple clips by performing motion inbetweening. Equipped with this ability, future work in embodied agents can be paired with a natural language processor to perform complex tasks. Our proposed PMCP can be used as a general framework to enable progressive RL and multi-task learning. In addition, we show that one can use **only 3D keypoint** as motion input for imitation, alleviating the requirement of estimating joint rotations. Essentially, we use PHC to perform inverse kinematics based on the input 3D keypoints and leverages the laws of physics to regulate its output. We believe that PHC can also be used in other areas such as embodied agents and grounding, where it can serve as a low-level controller for high-level reasoning functions.

References

[1] Jinkun Cao, Xinshuo Weng, Rawal Khirodkar, Jiangmiao Pang, and Kris Kitani. Observation-centric SORT: Rethinking SORT for robust multi-object tracking. Mar. 2022.

[2] Leonard Hasenclever, Fabio Pardo, Raia Hadsell, Nicolas Heess, and Josh Merel. CoMic: Complementary task learning & mimicry for reusable skills. http://proceedings.mlr.press/v119/hasenclever20a/hasenclever20a.pdf. Accessed: 2023-2-13.

[3] Jocher, Glenn and Chaurasia, Ayush and Qiu, Jing. Yolov8.

[4] Jiefeng Li, Chao Xu, Zhicun Chen, Siyuan Bian, Lixin Yang, and Cewu Lu. HybrIK: A hybrid analytical-neural inverse kinematics solution for 3D human pose and shape estimation. Nov. 2020.

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

[6] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. MCP: Learning composable hierarchical control with multiplicative compositional policies. May 2019.

[7] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. AMP: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, (4):1–20, Apr. 2021.

[8] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv [cs.LG]*, June 2016.

[9] István Sárándi, Timm Linder, Kai O Arras, and Bastian Leibe. MeTRAbs: Metric-scale truncation-robust heatmaps for absolute 3D human pose estimation. *arXiv*, pages 1–14, 2020.

[10] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Amit H Bermano, and Daniel Cohen-Or. Human motion diffusion model. *arXiv [cs.CV]*, Sept. 2022.

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