Locomotion-Action-Manipulation: Synthesizing Human-Scene Interactions in Complex 3D Environments (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 shows the results of our method, LAMA, on various scenarios. In the video, we show our human motion synthesis results on PROX [9], Matterport3D [4], and also our own 3D scene scanned by Polycam App [1] with an iPad pro. We use SAPIEN [20] object meshes to semi-automatically produce manipulation cues, which is also shown in our videos. As shown, our method successfully produces plausible and natural human motions in many challenging scenarios.

While our original pipeline is designed for test-time optimization, in our video we also qualitatively demonstrate the strength of our framework in generalized scenarios by using a single optimized policy in handling different inputs without further optimization. In the video, we also show a policy optimized via our augmentation strategy (Sec. 3.5.) can handle more extensive input variations.

Our supplementary video contains several ablation studies of our method by showing the importance of collision reward r_{coli} in Eq. (4), transition reward ($r_{\Delta \text{t}}$, $r_{\Delta \text{v}}$) in Eq. (8), posture offset $\mathbf{a}_t^{offset}$ in Action Controller (Sec. 3.2), and our motion editing modules (Sec. 3.5) compared to the traditional Inverse Kinematics (IK). We also show the comparison with previous state-of-the arts [8, 18, 19] and demonstrate that our results produce better quality motions with improved collision avoidance performance in complex 3D scenes.

2. More Details on Experiments

In this section, we describe further details on our experiments on the Robustness Test of Action Controller (in Sec. 4.4) and the Perception Study (in Sec 4.2).

2.1. Robustness Test of Action Controller (Sec. 4.4)

In Sec. 4.4, Fig 10, and Table 2 of our main paper, we demonstrate that a single policy optimized for a specific input can handle varying target actions ϕ_A and initial $\mathbf{g}_{init}$.

Name	Foot Contact	Penetration
Single Optimized Policy	4.16	1.35
Generalized Policy	4.47	1.41

Table 1: Physical plausibility measurement of the synthesized motion from the robustness test. (Sec 4.4)

We describe more details on the experiment in Sec 4.4. For the experimental setup, we consider all possible variations for the input to test the generalization ability of the policy trained to a specific input. Specifically, a set of "all" valid initials $\{\mathbf{g}_{init}\}$ is automatically chosen via grid sampling of the floor plane for the locations $\mathbf{p}^0_{root}$, by excluding points occupied by objects, with a random body orientation for $\mathbf{r}^0_{root}$. For the action target $\{\phi_A\}$, we manually choose multiple plausible locations (e.g., chairs) for the actions. In the test scene $\mathbf{W}_0$ we use in Fig. 10, there exist 2635 plausible initial positions and we consider 4 target action cues shown in the white boxes in Fig. 10.

The original policy π_o (Fig. 10 top) is optimized to a specific input $\mathbf{g}_{init}$ and action cue ϕ^1_A, marked as red in the top left of Fig. 10. The colored points in Fig. 10 show the locations where the policy π_o achieves the goal successfully without any further optimization for the policy. For each input pair $\mathbf{g}_{init}$ and ϕ^n_A, we perform the motion synthesis with the policy π_o 5 times. In each trial, the initial body orientation is chosen randomly to provide more variations. We determine the policy is successful for the current initial location when no early termination conditions (collision, stall, moving out of the scene) are met while fulfilling ϕ^n_A at least twice out of the 5 trials. As shown in Fig. 10 and Table 2, our action controller optimized for a specific target can be applicable to many input variations.

We perform the same test for the generalized policy (described in Sec. 3.5) in the bottom of Fig. 10 and Tab. 2. As shown, this policy can cover much more extensive input variations on the same scene.

Comparison of Computation Time. As a test-time optimization without requiring scene-paired motion datasets, our original framework takes time to train a policy from a scratch for a given input pair. However, reusing the same policy that is optimized for the specific input for other inputs can greatly reduce the computation time, because no further optimization is needed for the policy. To compare the time between performing the inference only and optimizing a policy from scratch, we test with 5 input pairs consisting of initial $\mathbf{g}_{init}$ and ϕ_A. Here, the term "inference only" indicates that we use a pre-optimized policy without any further optimization for varying inputs. As the result, the inference-only scenario takes **0.15** seconds on average per input pair for motion synthesis, while optimizing a policy from scratch per pair takes **6.32** minutes (379 seconds) on average. As shown, the capability of the reinforcement learning framework provides the potential to greatly improve the efficiency of our method.

Motion Quality Measurement. We also evaluate the physical plausibility of the synthesized motion in the robustness test in Sec. 4.4. An optimized policy synthesized 15 motion sequences with distinct input pairs (the input pair which the policy is initially optimized to is not included). We also perform the measurement to motions synthesized by the generalized policy optimized with an augmentation strategy. The results are shown in Table 1. This shows while a policy can handle variations in input, there is no performance drop in the synthesized motion quality.

2.2. Perception Study Setup

The videos used for perception study are in the supplementary video. We include 3 videos per set to the supplementary video.

3. More Details on Implementations

3.1. Action Controller

Implementation Details. The policy and the value network of the action controller module consists of 4 and 2 fully connected layers of 256 nodes, respectively. The control policy is optimized through Proximal Policy Optimization (PPO) algorithm [17]. Adam optimizer [10] is used with Nvidia RTX 3090 GPU. For the action controller $\mathbf{A}$ and motion synthesizer module $\mathbf{S}$, we use the animation library DART [11]. We also use a publicly available PPO implementation [14, 12], where we remove the variable time-stepping functions stepping in [12] by following the original PPO algorithm. The details of the optimization regarding the policy and value network of the action controller are written in Table 2.

Acceleration Techniques. As written in the main paper, we use early termination conditions to accelerate policy optimization. The episode is terminated when (1) the character moves out of the scene bounding box; (2) when the collision reward r_{coli} is under a certain threshold; and (3) the root velocity for a specific time duration (50 frames) is under a certain threshold to prevent the character standing still for a overly long time. Also, the action controller first checks in advance whether the action signal is valid when it makes transitions from locomotion to other actions. When the nearest feature distance of Eq. 2 in the motion synthesizer (Sec. 3.3) is over a certain threshold, the action controller discards the transition and continues navigating.

Name	Value
Learning rate of policy network	2e-4
Learning rate of value network	0.001
Discount factor (γ)	0.95
GAE and TD (λ)	0.95
Clip parameter (ϵ)	0.2
# of tuples per policy update	30000
Batch size for policy/value update	512

Table 2: Details on the hyper-parameters for learning the control policy of the Action Controller $\mathbf{A}$.

3.2. Motion Synthesizer

Motion Database Information. Motion is captured by an IMU based system XSens MVN Link [3] and is post-processed via XSens MotionCloud software [2]. The captured motion is then retargeted to a single unified skeleton using Autodesk MotionBuilder and is post-processed to be suitable for motion matching. For action motions we mirror the motion segments for data augmentation. The length (in frames) of motion segments ("Seg. Length" in tables), number of motion segment ("Seg. Count" in tables), and the number of total frames ("Total Frames" in tables) are summarized in Table 3.

Action-Specific Feature Definition. The motion feature, as defined in our main paper Sec 3.3, represents both the current state of the motion and a short term future movements: $f(\mathbf{m}) = \{\{p_{\text{j}}\}, \{\dot{p}_{\text{j}}\}, \theta_{up}, c, \mathbf{o}_{\text{future}}\}$. In particular the action specific feature $\mathbf{o}_{\text{future}} = \{\{p_0^{\Delta t}\}, \{r_0^{\Delta t}\}\}$ contains future motions so that the motion search process can take into account the future motion consistency, where $p_0^{\Delta t}, r_0^{\Delta t} \in \mathbb{R}^2$ are the position and orientation of root joint at Δt frames later from the current target frame. For locomotion, we extract $\Delta t = 10$, 20, and 30 frames in the future (at 30Hz) following [5], as addressed in our main paper. For sitting, we specifically choose Δt as the frame where the character completes the sit-down motion. The major motivation of this design choice is encourage the motion synthesizer to search the motion clips with the desired target

action.

3.3. Motion Editing via Motion Manifold

Implementation Details for Models and Training. The encoder and decoder of the task-adaptive motion editing module consist of three convolutional layers. For the convolutional autoencoder of task-adaptive motion editing, we use PyTorch [15], FairMotion [6], and PyTorch3d [16]. The autoencoder is trained with the Adam optimizer [10] with learning rate 0.0001. We use Nvidia RTX 3090 GPU. We use 3 layers of 1D temporal-convolutions with kernel width of 25 and stride 2, and the channel dimension of each output feature is 256. For training the autoencoder module in task-adaptive motion editing we use data in Mixamo [13], Lafan1 [7], COUCH [21], and ours. The training datasets are summarized in Table 4. Note that data used for training the autoencoder also does not include scene related information (in bvh format), and we use different pre-processing steps between the Motion Editing module and the Motion Synthesizer.

Reconstruction Loss. The encoder Ψ and decoder Ψ^{-1} are trained based on reconstruction loss $\mathcal{L}_{\text{recon}} = ||\mathbf{X} - \Psi^{-1}(\Psi(\mathbf{X}))||^2$, where:

$$\mathcal{L}_{\text{recon}} = w_c\mathcal{L}_{\text{contact}} + w_r\mathcal{L}_{\text{root}} + w_q\mathcal{L}_{\text{quat}} + w_p\mathcal{L}_{\text{pos}}. \quad (1)$$

$\mathcal{L}_{\text{contact}}$, $\mathcal{L}_{\text{root}}$, and $\mathcal{L}_{\text{quat}}$ are the MSE losses of foot contact labels, root status (height and transform relative to the previous frame projected on the XZ plane), and the joint rotations in 6D representations [22]. To penalize errors accumulating along the kinematic chain, we perform forward kinematics (FK) and measure the global position distance of joints between the original and reconstructed motion. As global positions of the joints are highly dependent on the root positions, for the early epochs, the distance is measured based on root-centric coordinates to ignore the global location of roots, which we found empirically more stable. Also, during training we used an augmentation technique of adding noise to normalized input. Noise is sampled from the normal distribution $\mathcal{N}(0,\ 1)$ multiplied with a scale of 0.01. We found the technique empirically increases reconstructed motion quality.

Motion Editing Loss. For motion editing, the positional loss and regularization loss are defined as follows.

$$\begin{aligned}
\mathcal{L} &= w_p\mathcal{L}_{\text{pos}} + w_f\mathcal{L}_{\text{foot}} + w_r\mathcal{L}_{\text{root}}, \quad \text{where} \\
\mathcal{L}_{\text{pos}} &= \sum_{\mathbf{q}_j^t \in \phi_M} \|\mathbf{p}_j^t - \mathbf{q}_j^t\|^2, \\
\mathcal{L}_{\text{foot}} &= \sum_{foot} \|\mathbf{p}_{foot}^e - \mathbf{p}_{foot}^i\|^2, \\
\mathcal{L}_{\text{root}} &= w_r\|\mathbf{r}_{\text{xz}}^e - \mathbf{r}_{\text{xz}}^i\|^2 + w_{\Delta r}\|\dot{\mathbf{r}}_{\text{xz}}^e - \dot{\mathbf{r}}_{\text{xz}}^i\|^2.
\end{aligned} \quad (2)$$

$\mathbf{p}_j$ denotes positions of joint j, and $\mathbf{r}$, $\dot{\mathbf{r}}$ denotes root positions and velocities respectively. Superscript e and i indicates whether it is from edited or initial motion, respectively. Subscript xz indicates the vector is projected onto the XZ plane. The loss term $\mathcal{L}$ enforces the edited motion to maintain contact and root trajectory (in the XZ plane) of the initial motion, while generating natural movements of the other joints to meet the sparse positional constraints. For minimizing losses, Adam optimizer [10] is used as well with a learning rate of 0.005.

Generating Manipulation Cues from SAPIEN [20]. While the manipulation cue $\phi_M = [\mathbf{v}(R_t, T_t, \theta_t)]_t$ can be provided via diverse ways depending on the applications, we mainly consider the scenarios of interacting with articulated objects. For this purpose, we semi-automatically produce the manipulation cues by extracting the desired target vertex trajectories of the parts of articulated objects from the SAPIEN dataset [20]. Specifically, we place a target object in our 3D scene, and choose a target vertex $\mathbf{v}$ of the object where we assume the character's hand contacts to manipulate the target part (e.g., a vertex in the lid of a trash can object). Then, the trajectory of the vertex $\phi_M = [\mathbf{v}(R_t, T_t, \theta_t)]_t$ can be obtained by varying the parameter for the articulated motion θ with a fixed interval, where R_t, T_t, are the global orientation and translation of the object and θ_t is the parameters for the object articulation (e.g., the hinge angle of the cover of a laptop) at time t. $\mathbf{v}(\cdot)$ represents the 3D location of the chosen vertex $\mathbf{v}$ given the parameters. The resulting manipulation cue ϕ_M is the target trajectory that a hand joint should follow for the manipulation motion. Note that our system requires only the manipulation cue ϕ_M, and the 3D object mesh is shown only for visualization purposes, where we visualize it with the synced θ_t.

Further Implementation Details for Manipulation Given the initial motion output $\mathbf{M}$ synthesized from the Action Controller, $\mathbf{M} = \{\mathbf{m}_t\}_{t=1}^{T}$ and the manipulation cue $\phi_M = \{\mathbf{m}_{t'}\}_{t'=1}^{\tau}$, our system is also given the corresponding time segment $[t_i, t_f]$ where we want to edit the motion to follow the manipulation cue (we assume the same duration, i.e., $t_f - t_i = \tau$). Then motion editing is performed on the target motion segment, $\tilde{\mathbf{M}}_{t_i:t_f} = \mathbf{E}(\mathbf{M}_{t_i:t_f})$, which subsequently replaces the corresponding part in $\mathbf{M}$ to form $\tilde{\mathbf{M}}$ as the final output.

Depending on possible applications (e.g, sitting down and opening a laptop), the manipulation motion may need to be "added" in the middle or the end of the synthesized motion $\mathbf{M}$. In this case, we simply duplicate the target frame by τ to build a longer motion $\mathbf{M} = \{\mathbf{m}_t\}_{t=1}^{T+\tau}$, and apply the motion editing to the target motion segment that is a stationary motion produced via the duplication.

Label	Seg. Length	Seg. Count	Total Frames
Locomotion	10	23832	24267
Sit	50 – 85	6230	15130

Table 3: Details on pre-processed motion datasets per each action category of the motion database in $\mathbf{S}$.

Name	Value
Motion sequence length	120
Number of sequence (training)	45713
Number of sequence (test)	11040
Number of sequence (validation)	5268

Table 4: Details on pre-processed motion datasets for training our motion editing module $\mathbf{M}$.

References

[1] Polycam - lidar and 3d scanner for iphone android. https://poly.cam/. 1

[2] Xsens motioncloud. https://www.movella.com/products/motion-capture/xsens-motioncloud. 2

[3] Xsens mvn link. https://www.movella.com/products/motion-capture/xsens-mvn-link. 2

[4] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *3DV*, 2017. 1

[5] Simon Clavet. Motion matching and the road to next-gen animation. In *Proc. of GDC*, 2016. 2

[6] Deepak Gopinath and Jungdam Won. fairmotion - tools to load, process and visualize motion capture data. Github, 2020. 3

[7] Félix G Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening. *ACM Trans. Graph*, 39(4), 2020. 3

[8] Mohamed Hassan, Duygu Ceylan, Ruben Villegas, Jun Saito, Jimei Yang, Yi Zhou, and Michael Black. Stochastic scene-aware motion prediction. In *ICCV*, 2021. 1

[9] Mohamed Hassan, Vasileios Choutas, Dimitrios Tzionas, and Michael J. Black. Resolving 3D human pose ambiguities with 3D scene constraints. In *ICCV*, 2019. 1

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2, 3

[11] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22), 2018. 2

[12] Seyoung Lee, Sunmin Lee, Yongwoo Lee, and Jehee Lee. Learning a family of motor skills from a single motion clip. *ACM Trans. Graph.*, 40(4), 2021. 2

[13] Adobe's Mixamo. https://www.mixamo.com, 2017. 3

[14] Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.*, 38(6), 2019. 2

[15] 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. 3

[16] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 3

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2

[18] Jingbo Wang, Yu Rong, Jingyuan Liu, Sijie Yan, Dahua Lin, and Bo Dai. Towards diverse and natural scene-aware 3d human motion synthesis. In *CVPR*, 2022. 1

[19] Jiashun Wang, Huazhe Xu, Jingwei Xu, Sifei Liu, and Xiaolong Wang. Synthesizing long-term 3d human motion and interaction in 3d scenes. In *CVPR*, 2021. 1

[20] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *CVPR*, 2020. 1, 3

[21] Xiaohan Zhang, Bharat Lal Bhatnagar, Sebastian Starke, Vladimir Guzov, and Gerard Pons-Moll. Couch: Towards controllable human-chair interactions. In *ECCV*, 2022. 3

[22] Yi Zhou, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. On the continuity of rotation representations in neural networks. In *CVPR*, 2019. 3