SMGDiff: Soccer Motion Generation using diffusion probabilistic models

Supplementary Material

In this supplementary, we first provide a detailed exposition of the implementation specifics of our proposed method. We then describe the data capture process and present additional results from our dataset. Finally, we showcase further qualitative results to demonstrate the effectiveness of our approach.

1. Methods Details

1.1. Trajectory Generation Model

Network Architecture. We employ a transformer model with 8 encoder-only layers, each comprising 4 attention heads, to learn the mapping from user input to diverse trajectories. The input dimension of the transformer is set to 256 with the feedforward dimension set to 1024 and the dropout rate set to 0.1. Following the *Seperate Condition Tokenization(SCT)* method proposed in CAMDM [1], we first use separate tokenizers for each input condition and then concatenate the condition tokens with the noisy motion sequence. Specifically, we use linear layers to transform the soccer skill label $\mathbf{S} \in \mathbb{R}^1 \mapsto \mathbb{R}^{256}$, target trajectory point $\mathbf{G} \in \mathbb{R}^8 \mapsto \mathbb{R}^{256}$ and past trajectory $\mathbf{T}^{\mathcal{P}} \in \mathbb{R}^{\mathcal{P}\times 8} \mapsto \mathbb{R}^{\mathcal{P}\times 256}$. After concatenating with the noise, the final input dimension is $\mathbb{R}^{(\mathcal{P}+\mathcal{F}+2)\times 256}$. During the forward pass, we add the input with a standard positional embedding. The encoded features are then processed through the transformer encoder and the trajectory output $T \in \mathbb{R}^{\mathcal{F}\times 256} \mapsto \mathbb{R}^{\mathcal{F}\times 8}$ is obtained via a final linear layer.

Training. In training, the reconstruction loss and velocity loss are specifically defined as follows:

$$\mathcal{L}_{\text{recon}} = \frac{1}{F}\sum_{i=1}^{F} ||z_0^i - \hat{z}_0^i||_2^2, \tag{1}$$

$$\mathcal{L}_{\text{vel}} = \frac{1}{F-1}\sum_{i=1}^{F-1} ||(z_0^{i+1} - z_0^i) - (\hat{z}_0^{i+1} - \hat{z}_0^i)||_2^2, \tag{2}$$

where z_0 represents the groundtruth trajectory extracted from the training data and $\hat{z}_0$ represents the model predicted future trajectory, superscript i represents the frame index. The two training losses share the same weight $\lambda_{recon} = \lambda_{vel} = 1$. During training, we optimize the model using the AdamW optimizer [2] with a weight decay of 0.01 and a learning rate of 10^{-4}. Training is conducted for 4000 epochs on a single NVIDIA GeForce RTX 4090 GPU, with the batch size set to 512.

1.2. Soccer Motion Generation Model

Network Architecture. Our soccer motion generation model adopts a transformer-based architecture with 4 encoder-only layers, following a structure similar to the trajectory generation model described in Sec. 1.1. We use linear layers to transform the soccer skill label $\mathbf{S} \in \mathbb{R}^1 \mapsto \mathbb{R}^{256}$, past soccer motion $\mathbf{X}^{\mathcal{P}} \in \mathbb{R}^{\mathcal{P}\times(28\times 6)} \mapsto \mathbb{R}^{\mathcal{P}\times 256}$ and future trajectory $\mathbf{T}^{\mathbb{R}^{\mathcal{F}\times 8}} \mapsto \mathbb{R}^{\mathcal{F}\times 256}$. Note that here we map the human root position $h_p \in \mathbb{R}^3 \mapsto \mathbb{R}^6$ and ball state $b \in \mathbb{R}^{3+3+1} \mapsto \mathbb{R}^{6+6+6}$ using zero padding. We then concatenate the human root position and ball state with human joint rotation $h_\theta \in \mathbb{R}^{24\times 6}$ to get the soccer motion $X \in \mathbb{R}^{28\times 6}$, which serves as the input and output to the network.

Training. During training, the loss weights are set to $\lambda_{pos} = \lambda_{vel} = \lambda_{foot} = 1$. The model is optimized using the AdamW optimizer [2] with a weight decay of 0 and a learning rate of 5×10^{-4}. Training is conducted for 1000 epochs on a single NVIDIA GeForce RTX 4090 GPU, with a batch size of 512.

1.3. Contact Guidance Module

Implementation details. In the demo presentation, we adjust the activation of the contact guidance module. Contact Guidance is only applied during dribbling, trick or shooting. We also prevents misuse of the contact guidance module when the human-ball distance exceeds the control radius of $r = 2m$. In such cases, the character transitions to an off-the-ball move style, with the ball's position updated via physical simulation. We use relative representation for soccer ball to prevent excessive displacement between the character and the ball during dribbling.

Hyperparameters. In the contact guidance module, several hyperparameters are introduced, including the acceleration threshold and the penalty term for a lifted foot. This section details the methodology used to determine these hyperparameters. During the data preprocessing stage, ball-foot contacts within the dataset are annotated. Initially, a subset of contact instances is manually labeled. Subsequently, a statistical analysis is conducted to establish distance thresholds for ball-foot contact, as defined by the following equation:

$$c_b^j = \mathbb{I}((b_p - f_p^j) \leq \tau_d), \tag{3}$$

where c_b^j represents the contact between the ball and the joint, superscript j denotes the joint index, b_p is the global ball position, f_p^j is the global foot joint position, and $\tau_d = 0.1m$ is the distance threshold. Based on the distance-based ball-foot contact annotations, we further determine contacts using

Figure 1. Motion capturing setup, the blue cameras are OptiTrack Prime x13 cameras for capturing soccer motion.

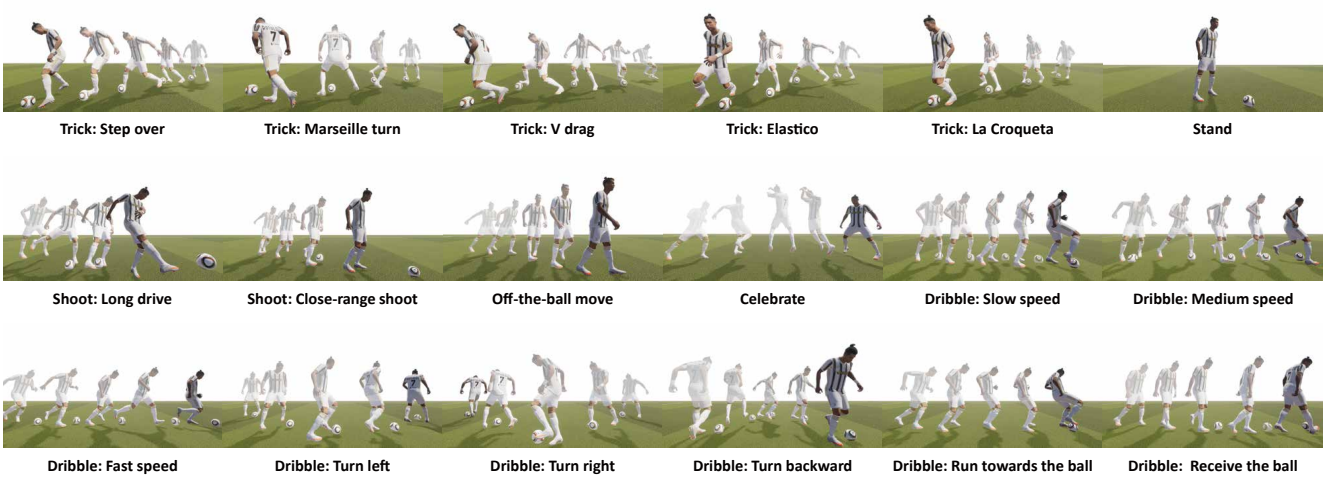

Figure 2. More results of our dataset. We provide detailed descriptions for each action category.

τ_a	$0.5m/s^2$	$1m/s^2$	$1.5m/s^2$	$2m/s^2$	$3m/s^2$
Acc.↑	56.66%	67.13%	79.12%	**91.47%**	83.31%

Table 1. Accuracy of different acceleration thresholds.

w_d	1	1.5	2	2.5	3
Acc.↑	88.31%	89.19%	**94.43%**	91.41%	87.01%

Table 2. Accuracy of different penalty terms.

the ball's acceleration. The accuracy of contact determination is evaluated under various acceleration thresholds τ_a, as shown in Tab. 1. A threshold that is too low leads to unintended contact enforcement, causing the foot and ball to stick together continuously. Conversely, an excessively high threshold eliminates meaningful ball-foot contact, rendering this module ineffective. Based on these observations, the acceleration threshold is ultimately set to $2m/s^2$. For the penalty term w_d, we conduct a statistical analysis on the dataset. The dataset includes annotations indicating which joint contacts the ball, as determined by Eq. 3. By applying the acceleration threshold, we calculate the required contact duration for the ball and identify the specific joint responsible for contact based on distance, with accuracy results presented in Tab. 2. When the penalty term is set to 1, it causes the ball to stick to the foot when the distance between the foot and the ball is very small. Conversely, an excessively large penalty term prevents the grounded foot from making contact with the ball. Ultimately, we set the penalty term to 2. The hyperparameters can be fine-tuned based on factors such as the friction of the environment and the size of the ball to ultimately achieve the most effective performance.

1.4. Implementation Details

Runtime Analysis. We provide a detailed analysis of the efficiency of each component. In the trajectory generation stage, the single-step diffusion model deployed in the Unity engine, combined with the blending strategy, takes approximately 8ms. In the soccer motion generation stage, due to the addition of 2 contact guidance steps in the diffusion process, the computation of the loss function and backward pass increases the inference time. Consequently, our 8-step denoising diffusion model takes approximately 80ms in total. Lastly, the network transmission time from Python to Unity is about 7ms. Overall, our method takes approximately 95ms for the whole inference process. Future work could explore reducing the computation time for contact guidance to further optimize the efficiency of the algorithm.

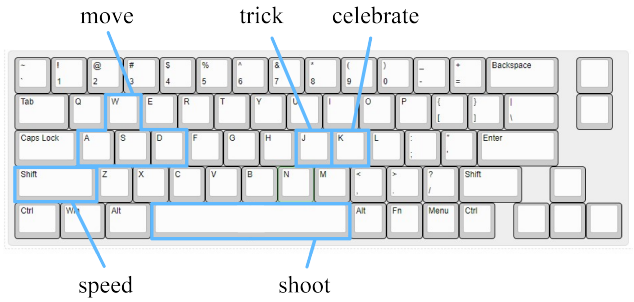

Figure 3. Keyboard control details.

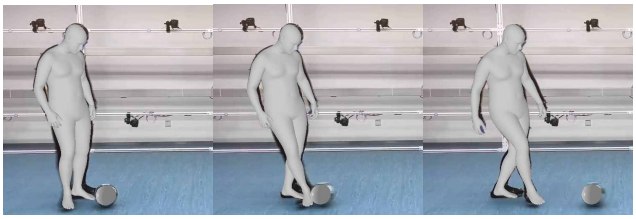
Figure 4. Qualitative evaluation of our dataset.

Runtime environment. For the environmental setup, we implement real-time character control through network communication between the Unity engine and Python. Initially, Unity captures user keyboard inputs and generates future trajectories using the employed trajectory generation model. The trajectory conditioning is then transmitted to Python via a TCP network. Utilizing the soccer motion generation model, the Python side generates human translation, joint rotations, and ball translation. The outputs are subsequently sent back to Unity for visualization, with the ball's rotation computed using Unity's physics engine. All model inferences and network communications are executed on a single machine equipped with an Intel Core i7-10700K CPU and an NVIDIA GeForce RTX 3080 Ti GPU.

Keyboard control. Fig. 3 illustrates the details of keyboard controls for soccer movement. The 'WASD' keys control direction, while 'Shift' accelerates speed. For style transitions, 'J' activates trick skills, 'Space' is used for shooting, and 'K' triggers celebrations. The transition between dribbling and off-the-ball movement is determined by the ball-foot distance: if it is less than the control radius $r = 2$m, the character will dribble; otherwise, it will perform an off-the-ball move.

2. Dataset Details

Capture setting. We provide the capture details of the Soccer-X dataset. During motion capturing, players wore motion capture suits equipped with 41 markers, leveraging the baseline skeletal framework provided by Motive software [3]. The soccer ball was asymmetrically marked with 12 markers to ensure accurate tracking and was defined as a rigid body. Our motion capture setup consists of 16 cameras arranged in a surrounding configuration. Fig. 1 shows the motion capture setup, where the blue cameras indicate those used during the capture process. The dataset includes annotations for action name and ball-foot contact events. As shown in Fig. 2, we also provide a detailed description of each action category.

Category	Property	Value
Football	Mass	0.43 kg
	Drag	0.2
	Angular Drag	0.05
	Radius	0.11 m
	Dynamic Friction	0.5
	Static Friction	0.5
	Bounciness	0.1
	Friction Combine	Multiply
	Bounce Combine	Maximum
Ground	Dynamic Friction	0.6
	Static Friction	0.6
	Bounciness	0.05
	Friction Combine	Average
	Bounce Combine	Minimum
Environment	Gravity	-9.81 m/s² (Y-axis)
	Fixed Timestep	0.01 s

Table 3. Physics settings for simulating soccer ball movement in Unity.

Dataset evaluation. We validated our dataset using a subset with manually labeled 2D keypoints, triangulated for 3D joint coordinates as the standard. Our capture results achieve a MPJPE of 35.4 mm, a PA-MPJPE of 30.7 mm, and a ball-foot distance error of 28.1 mm. Fig. 4 illustrates a qualitative evaluation of our dataset, demonstrating our capture accuracy.

Physics simulation. For the shooting motion, due to field constraints, we were only able to record the trajectory data of the soccer ball during the first half of its movement. To address this limitation, we extracted the position and rotation information from the last two frames of the recorded data and calculated the linear velocity and angular velocity of the soccer ball in the final frame. Using these initial conditions, along with other necessary physical parameters, we simulated the second half of the trajectory in the Unity engine. The specific parameter settings are shown in Tab. 3.

In this context, Football is a GameObject with a Rigid-

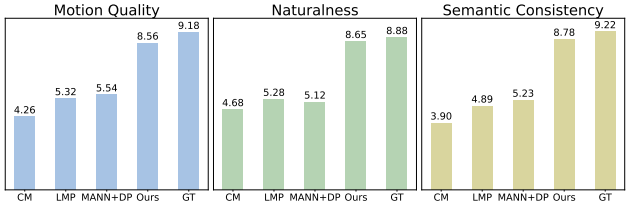

Figure 5. User study.

body component and a Sphere Collider component, Ground is a GameObject with a Collider component, and Environment refers to Unity's environment settings. The mass and radius of the football were set strictly according to the standards of the Fédération Internationale de Football Association (FIFA) and were consistent with the actual mass and dimensions of the soccer ball used in the experiment. Other parameters (such as friction coefficients and elasticity coefficients) were fine-tuned through multiple tests to ensure the simulation results closely matched with the real-world motion behavior.

In the demo, when the character's action corresponds to an off-the-ball movement style, the soccer state is updated using the same simulation method.

Dataset release clarification. The processes of data collection, capturing, labeling, and manual inspection for our dataset have been fully completed. We commit to open-sourcing all of our data. **We will release the dataset within one week of the paper's acceptance!**

3. More Experiment

User Study We invited 40 soccer enthusiasts to participate in a user study. The questionnaire consisted of multiple soccer motion sequences, including those generated by our method, the baseline methods, and the ground truth from the dataset. For each motion sequence, participants were required to rate three criteria on a scale from 0 to 10. Three criteria including motion quality, naturalness, and semantic relevance. The user study was distributed via Google Forms, with the order of motion sequences randomized to mitigate ordering bias. As shown in Fig. 5, both the ground truth and our method received favorable evaluations from most participants.

More Results. We further validate the diversity of the soccer motions generated by our method. As shown in Fig. 6, under identical user control input(including the same character speed, direction, and soccer skill label), our method generates soccer motions that exhibit different dribbling styles. This demonstrates that our trajectory generation model and soccer motion generation model have learned diverse character trajectories and soccer skills from our training data.

More Comparisons. We further compare our method with baseline approaches. Specifically, we use a complete sequence of the step-over skill in the trick category for a clearer comparison. As shown in Fig. 7, the soccer motions generated by LMP [4] and MANN-DP [5, 7] exhibits unnatural ball-foot contacts, while CM [6] fails to generate the corresponding step-over skill. In contrast, our method produces realistic and natural step-over motions.

References

[1] Rui Chen, Mingyi Shi, Shaoli Huang, Ping Tan, Taku Komura, and Xuelin Chen. Taming diffusion probabilistic models for character control. In *SIGGRAPH*, 2024. 1

[2] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017. 1

[3] Optitrack. NaturalPoint, Inc. Motion Capture Systems. https://optitrack.com/, 2022. 3

[4] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local motion phases for learning multi-contact character movements. *ACM Trans. Graph.*, 39(4), 2020. 4

[5] Sebastian Starke, Ian Mason, and Taku Komura. Deepphase: Periodic autoencoders for learning motion phase manifolds. *ACM Transactions on Graphics (ToG)*, 2022. 4

[6] Sebastian Starke, Paul Starke, Nicky He, Taku Komura, and Yuting Ye. Categorical codebook matching for embodied character controllers. *ACM Trans. Graph.*, 2024. 4

[7] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. Mode-adaptive neural networks for quadruped motion control. *ACM Trans. Graph.*, 2018. 4

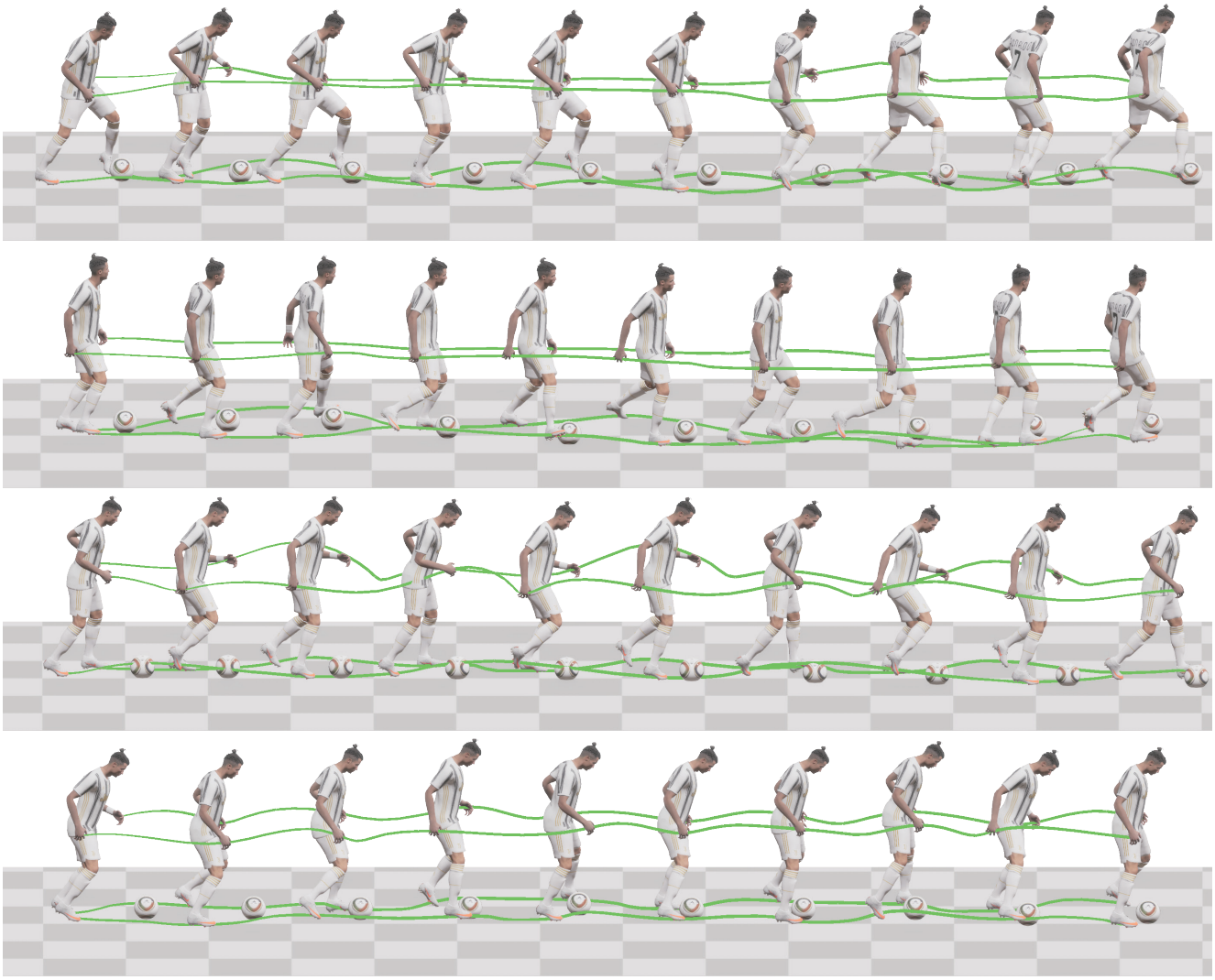

Figure 6. Additional qualitative results of SMGDiff. Given the same user control input, SMGDiff generates diver soccer motions.

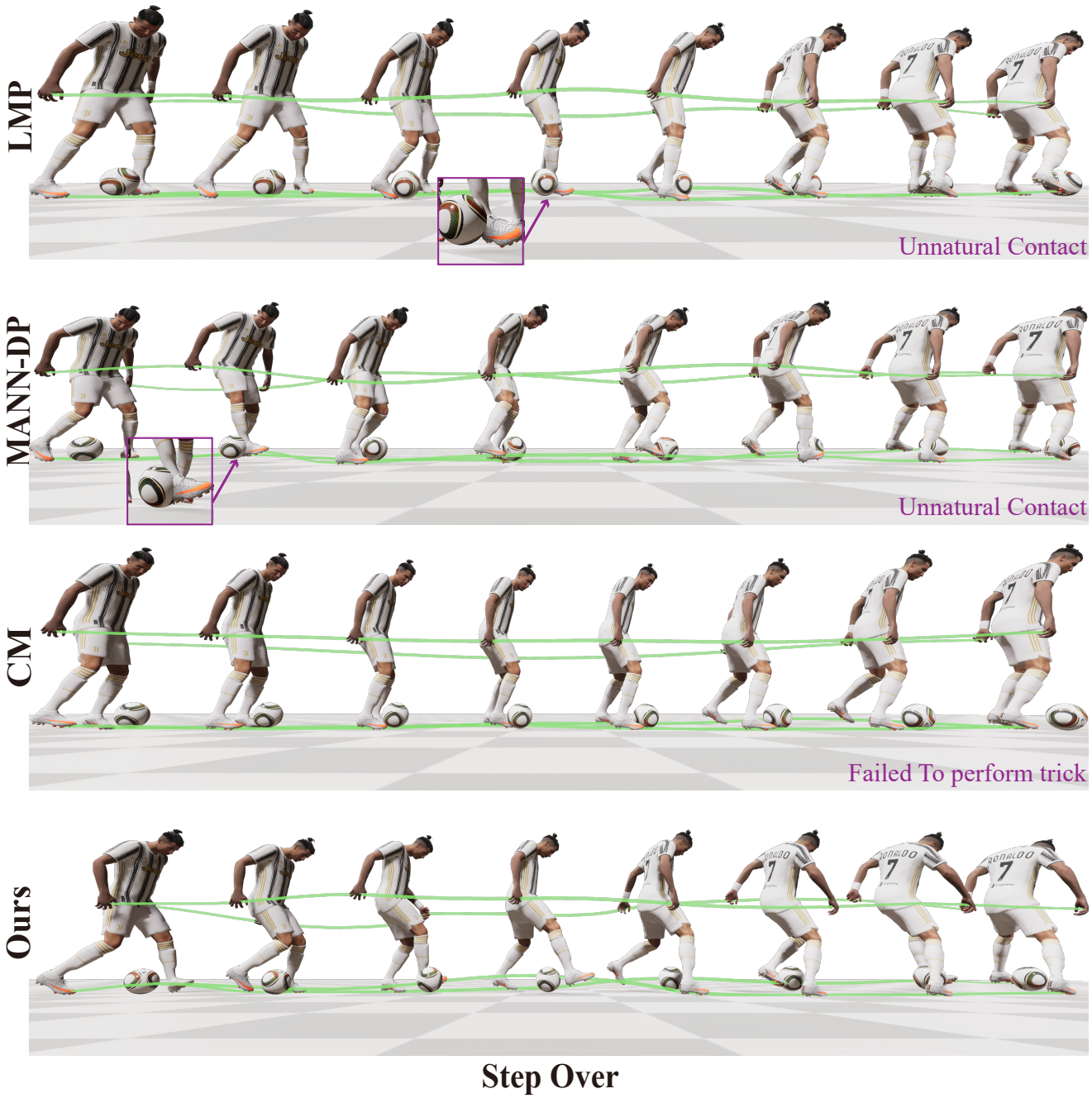

Figure 7. Additional qualitative comparisons. SMGDiff outperforms baselines on the step-over trick.