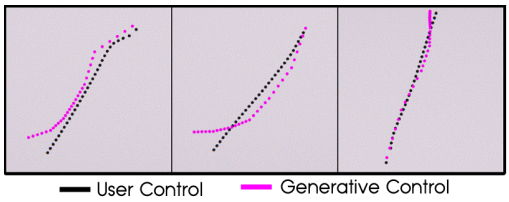

Fig. 7. Example inputs/outputs of the generative control scheme.

noise samples have variation with fine details. Using these as input to the motion prediction network will result in different body motion as we present in Section 9.

The question that arises is that of how to manage the controllability of the character when adding noise in the latent space as we do not condition the noise on the user instruction (see Section 10 for discussions about the design). We find the following approach to be simple and effective. We sample noise vectors from different random seed and use them as style variables. The animator can pick them as the behavior style of each character. For increasing the stochasticity, we can blend in and out the noise produced from such seeds selected randomly during runtime.

We also evaluate how the motion deviates from the original motion when changing the level of noise amplitude to find a level that can produce variations but stays close to the original control signal. The plots of scaling the noise level and the resulting RMSE of the control signals, as well as the corresponding motions are shown in Fig. 8. When the noise level exceeds 1, which corresponds to the standard deviation of the latent code of the generative control, the synthesized motion starts to either deviate significantly from the control signal or result in a corrupted motion. We find a level around 0.25 to 0.75 is suitable for producing variations while keeping the motion under control.

7 NETWORK TRAINING

The training is done by first normalizing the input and the output of the full dataset by their mean and standard deviation, pretraining the generative control model and then training the main network composed of the gating network and motion prediction network in an end-to-end fashion. The processed data is exported from Unity and used for training our framework implemented in TensorFlow.

For training the main network, similar to [Starke et al. 2019; Zhang et al. 2018], we use the AdamWR optimizer with the warm restart technique; the learning rate is initialized with the value of $1.0 \cdot 10^{-4}$ and later adjusted by the weight decay rate with the initial value of $2.5 \cdot 10^{-3}$. Dropout rate is set to 0.7, hidden layer size in the gating network is set to 128 and in the motion prediction network to 512 respectively.

Since we predict the local motion phase updates for the next and future frames to autoregress them back into the gating network input, we use teacher force training [Li et al. 2017; Williams and Zipser 1989] to make the network adapt to its own prediction inaccuracies. We slice each clip into sequences of 8 frames and use a uniform probability of 0.5 to update the phase input of the next frame by the output of the previous phase updates by Eq. (2). This helps to prevent the network from overfitting to future phase alignments that it would see during training but not during inference.

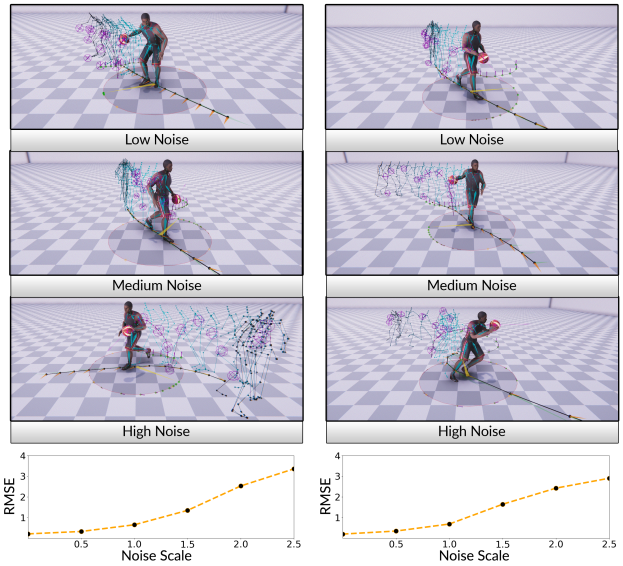

Fig. 8. Impact of noise scale on motion variation and controllability. Note that when the noise level is high, the character starts to deviate from the original control signal. Here the character is dribbling in a circle (left) or holding the ball while running (right).

Table 1. Motion capture data breakdown.

Label	Duration	Ratio
Total	211.6min	100%
Stand	39.1min	18.5%
Move	172.6min	81.4%
Dribble	115.9min	54.8%
Hold	12.7min	6.0%
Shoot	6.7min	3.2%

The composition of our complete dataset of 3 hours used for training is listed in Table 1, breaking down type and amount of clips and how we separate them into different actions. All motion is doubled by mirroring, downsampled from 60Hz capture to 30Hz, and then exported twice by shifting the data by one frame. After training, the data is compressed from ~3GB raw motion capture data (~8GB generated training data) to ~24MB network weights.

8 CHARACTER CONTROL SYSTEM

The character is controlled via a gamepad's joysticks and buttons (see Fig. 9) to offer a wide range of control signals to the user. The mapping between the user input to the actions are designed as follows. The translation and rotation motion are driven by the left joystick; when the joystick is simply tilted, the character moves to the direction without changing the orientation. When the joystick is rolled, its rotation is integrated overtime and the character turns its facing direction. When the joystick is pressed, the character

sprints. Left and right triggers are used for spin, which will not change the trajectory direction but selectively rotate the facing direction of the character in a quick fashion. The right joystick is mapped to the interaction vector $\mathbf{I}$, and used to control the dribbling locations around the character, which can produce movements such as switching the ball between hands, around the body or between legs. The dribbling height and frequency is controlled when the right joystick is pressed and moved horizontally or vertically. Pressing the Holding button will make the character stop dribbling and hold the ball, or catch the ball alternatively. Pressing the shooting button will make the character shoot the ball, where direction, height and speed is controlled by the interaction vectors. When the character is in the defending mode, pressing Attacking button will make the character steal the ball from the opponent and take control of it.

Each player controls its own character, however, the network instance can be shared between different characters which have the same network weights. This effectively saves memory and storage since we do not require training multiple networks for different roles or actions.

9 EXPERIMENTS AND EVALUATION

Our animation system is implemented in the Unity engine, and the neural network is queried through a socket interface to compute the character movements. We conduct our experiments on a MSI GT75 Titan gaming laptop with Intel i7-9750H processing cores and a NVIDIA GeForce RTX2080 GPU, requiring 2-4ms per frame for each character including user control processing, inference time and scene rendering. We run the animation at 30Hz framerate.

9.1 Animated Results

We animate the character through the gamepad using the interface described in Section 8. The example snapshots for various movements are shown in Fig. 10. Realistic movements that appear similar to the motion capture data with few artefacts can be synthesized

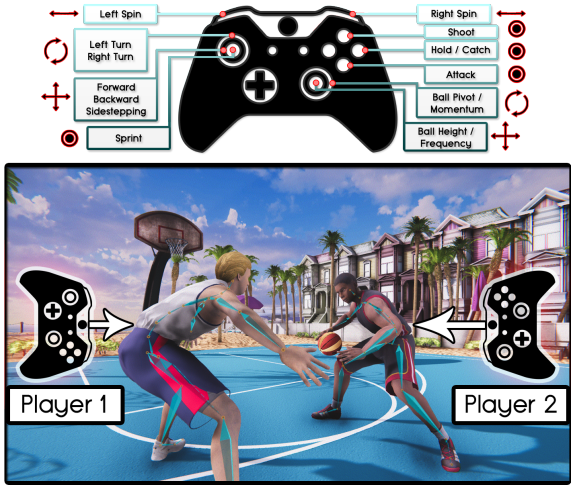

Fig. 9. Gamepad controls exposed to the user for directing a wide and continuous range of different character movements and actions.

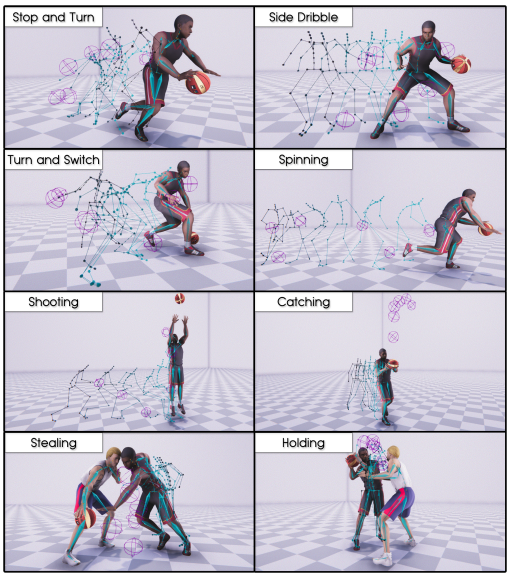

Fig. 10. Movement types that can be synthesized by our system.

in real-time. In addition to the complex character movements with multiple contacts and quick maneuvers, the interactions between multiple characters, such as stealing the ball and avoiding the opponent, can be produced. The interactions of two characters are produced by two players controlling each character via their own gamepad.

To animate the physically-plausible ball motion during catching and shooting movements, the ball motion is controlled via physical simulation. When catching the ball, we let the network override the simulated ball position and velocity with its predicted values based on the hand contact predictions. Similarly, the physics override the network predictions as the ball leaves the hands during shooting.

To animate stealing actions, we guide the hand of the character to the ball via inverse kinematics; this automatically activates the contact state of the hand and the ball and then the dribbling action automatically. As the system is trained with a variation of stealing motion, the corrupted motion by inverse kinematics are projected to natural stealing motion through this process.

The readers are referred to the supplementary video for the animated results.

9.2 Quantitative Evaluation

We provide a quantitative evaluation of our system in terms of the body movement, contact accuracy and responsiveness using a test dataset. The control variables of the test data are extracted as described in the Appendix, and given to our model as the input. Using the output motion we compute values for each criterion. We compare our method with other existing methods such as PFNN [Holden et al. 2017], MANN [Zhang et al. 2018] and LSTM [Lee et al. 2018]. We also note that it is in general difficult to directly compare complex animation systems, given that each system is a complex artefact unto itself.

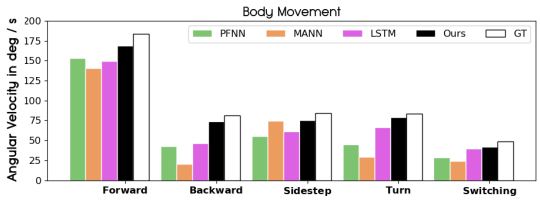

Fig. 11. Angular joint updates for different movements. Forward / Backward/ Sidestep: Medium-speed dribbling along straight line. Turn: Slow dribbling along a small circle. Switching: Dribbling the ball between hands while standing.

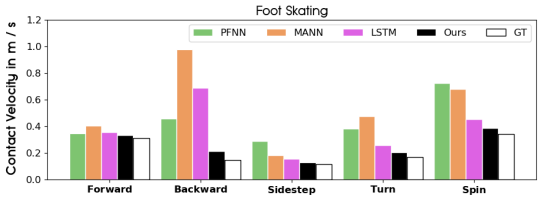

Fig. 12. Foot contact stability for different movements. Tested movements are same as for Fig. 11, with Spin: 360° rotation while running forward, which is a better motion for evaluating foot skating.

9.2.1 Body Movement. We quantify the body movement by the sum of the absolute angle updates of all the joints per frame. Motions synthesized by neural networks tend to smooth the motion; we found this metric provides a good indication of how agile and unsmoothed the generated motion appears. Comparing the values of our method with PFNN, MANN and LSTM in Fig. 11, our model outperforms all types of motion. The difference is more significant for movements where the data samples are rather sparse, such as backward and turning motion. There is much more data for forward motion where all the models train well, whereas our method does not overfit to those but can reconstruct the other motions too.

9.2.2 Contact Accuracy. We evaluate the contact accuracy between the feet and the ground. For that contact accuracy between the feet and the ground, we compute the amount of foot skating by summing the horizontal movements of the feet when their height and vertical velocity are below thresholds and thus should be in contact. As can be observed in Fig. 12, our method has the lowest foot skating. We also plot the distance between the hand and the ball during a dribbling motion in Fig. 13. To produce an accurate dribbling, the distance needs to go zero for every bounce of the ball; this happens only consistently for our model (black line) and LSTM (magenta line), whereas the latter produces occasional ball movement artefacts.

9.2.3 Responsiveness. The responsiveness of the character to the user inputs is one of the most important aspects in character control, and can be evaluated by measuring how much time is needed in average to reach the target speed and orientation, as well as for completing tasks such as spinning since the user input signal was given. The plot of the average time required for completing the tasks

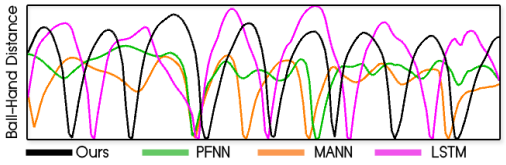

Fig. 13. The distance between the ball and the dribbling hand. Only our model produces a curve where the distance reaches zero at every bounce, while the other methods occasionally fail to bounce back to the hand.

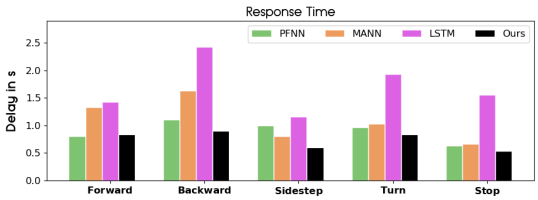

Fig. 14. Response time of character movements to the given user controls. Tested movements are same as for Fig. 11, with Stop: suddenly stopping from a running motion, which is a better motion to evaluate the response time.

are shown in Fig. 14. Our response is slightly worse than moving forward compared to PFNN though almost the same; for other movements our model outperforms the others. LSTM performs worse in all cases, possibly due to the lack of data augmentation. Our original dataset is already very large compared to those used for training other LSTM models, such as [Lee et al. 2018] (3 hours vs. 10-12min). Augmenting the data as done in [Lee et al. 2018] (2000% of the original data), may improve the responsiveness significantly, although will require much longer training time. Reducing the initial data size and then doing a data augmentation may be feasible, although that will result in less variation in the motion.

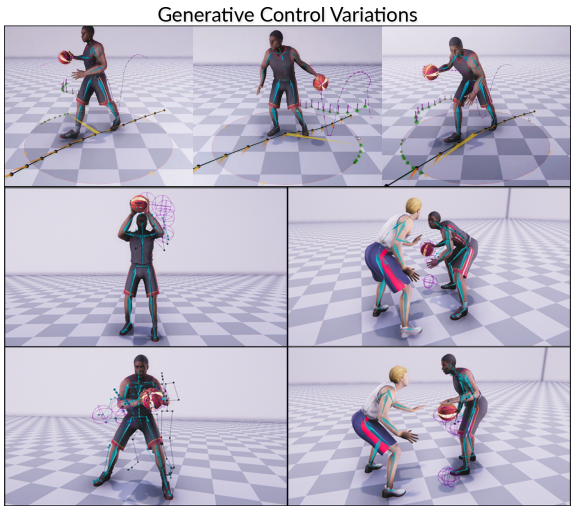

Fig. 15. Generated motion nuances from different noise seeds.