

Trajectory Augmentation for Robust Neural Locomotion Controllers

Dhruv Agrawal^{1,2}^a, Mathias König¹^b, Jakob Buhmann²^c, Robert Sumner^{1,2}^d
and Martin Guay²^e

¹*Department Informatik, ETH Zürich, Zurich, Switzerland*

²*DisneyResearch|Studios, Zurich, Switzerland*

dhruv.agrawal@inf.ethz.ch, matkoenig@gmx.ch, {jakob.buhmann, sumner,martin.guay}@disneyresearch.com

Keywords: Nearest Neighbor Search, Data Augmentation, Neural Networks, Motion Generation.

Abstract: Neural Locomotion Controllers are promising real-time character controllers that can learn directly from motion data. However, the current state of the art models suffer from artifacts such as Pose Blocking and Foot Skating caused by poor generalization to real world control signals. We show that this is due to training on unbalanced biased datasets with poor representation for many important gaits and transitions. To solve this poor data problem, we introduce Trajectory Augmentation (TrajAug), a fully automatic data augmentation technique that generates synthetic motion data by using motion matching to stitch sequences from the original dataset to follow random trajectories. By uniformly sampling these trajectories, we can rebalance the dataset and introduce sharper turns that are commonly used in-game but are hard to capture. TrajAug can be easily integrated into the training of existing neural locomotion controllers without the need for adaptation. We show that TrajAug produces better results than training on only the original dataset or a manually augmented dataset.

1 INTRODUCTION


Neural Locomotion Controllers aim to replace the complex animation blend trees that are currently used by most game titles. In place of defining rules and state transitions between different animation clips when using blend trees, Neural Locomotion Controllers can directly generate motion given user input. While we have witnessed several Neural Locomotion Controllers such as (Zhang et al., 2018; Henter et al., 2020; Holden et al., 2020) emerge in recent years, their performance gets limited by the lack of the complete motion capture dataset. For example, using the public LaFan1 dataset (Harvey et al., 2020), and training the public Mode-adaptive Neural Network (MANN) architecture, results in foot skating and blocking artifacts, as shown in our video. Recording a high quality motion capture dataset requires large amounts of planning and it is easy to overlook certain transitions and gait types, while simultaneously oversampling others. This often results


in multiple visits to the motion capture studio, further adding to costs.


Even with a well captured dataset, human-recorded motion is often not as responsive as what games require. In video games (e.g. using motion matching), exaggerated character motions are synthesized on the fly by blending motion clips in order to provide a responsive experience to the user. Training only on human motion does not cover this space of faster-than-real turns.


Data Augmentation has been widely embraced in machine learning. While affine transforms and color shifts for images in Computer Vision are well known examples, other sub-fields such as Natural Language Processing and Computational Biology also rely on data augmentation to improve generalization for tasks with limited data. These include machine translation and sampling pretrained large language models for text samples (Li et al., 2022) and even using GANs to generate synthetic medical data (Chin-Cheong et al., 2019). However, for training Neural Locomotion Controllers, data augmentation has been limited to only mirroring the original motion capture.


In this paper, we introduce a fully automatic data augmentation technique we call Trajectory Augmentation (TrajAug). From a source locomotion dataset,

^a <https://orcid.org/0009-0005-0442-9781>

^b <https://orcid.org/0009-0002-4674-100X>

^c <https://orcid.org/0009-0008-3038-4881>

^d <https://orcid.org/0000-0002-1909-8082>

^e <https://orcid.org/0009-0002-7496-6185>

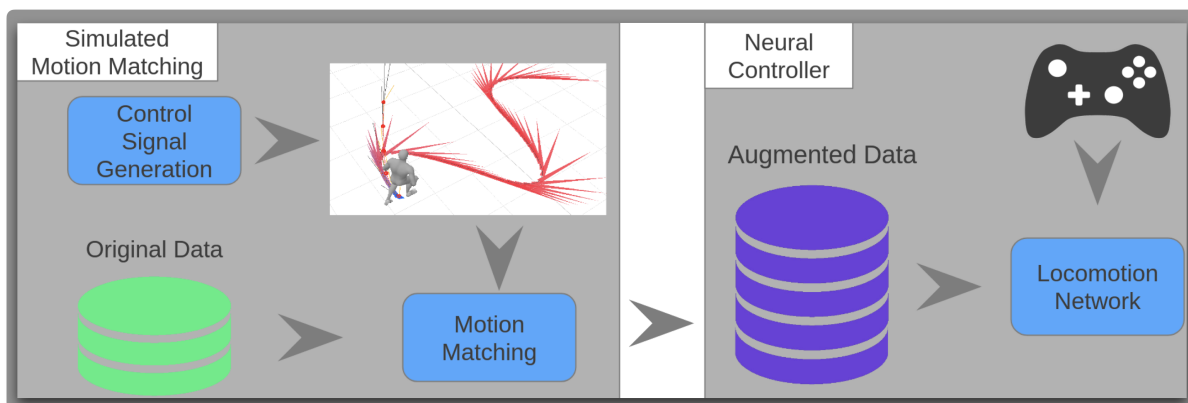


Figure 1: Trajectory Driven Augmentation takes a source mocap dataset and simulates user controls to synthesize a new dataset using motion matching. The augmented data is then used to train a neural controller with fewer artifacts (see our video) and is more responsive to real-time user directives.

we leverage motion matching to synthesize new motion clips in order to generate a balanced dataset that also includes sharper turns. Our approach can be easily incorporated into existing neural controller training without any modification and results in neural controllers with reduced artifacts, such as pose blocking and foot skating; and that is more responsive to user directives at run-time.

We show results of Trajectory Augmentation applied to the Mode-adaptive Neural Network (MANN) (Zhang et al., 2018) controller, as well as to the Learned Motion Matching (LMM) (Holden et al., 2020) controller, both trained on the LaFan1 dataset (Harvey et al., 2020). We evaluate these controllers for foot skating and responsiveness, and conclude that both qualitatively and quantitatively, Trajectory Augmentation improves the quality of neural controllers.

2 RELATED WORK

In practice, many interactive character controllers are based on blend trees that blend between motion clips (e.g. walking forward, left, right), and transition between different states such as walking, jumping, and idling. These controllers are very responsive due to the blending, but their authoring becomes increasingly complex with the number of actions and states the character can perform. Motion Matching (Clavet and Büttner, 2015) is a popular approach that operates on raw mocap data and uses nearest neighbor search, together with motion blending to synthesize movements on-the-fly corresponding to the user input. It has been used in popular video game titles such as *Fifa* and *Madden* to synthesize locomotion and character-scene interactions in real-time (Harrower, 2018; Allen, 2021). While motion matching is pop-

ular in practice due to the quality of movements and responsiveness, it keeps the dataset in memory which neural controllers—being compact learned models (Holden et al., 2017; Starke et al., 2020; Zhang et al., 2018; Starke et al., 2019; Ling et al., 2020; Holden et al., 2020; Lee et al., 2018)—promise to alleviate.

Neural Controllers. The first phase-functioned neural network (PFNN) (Holden et al., 2017) blends between four networks using a cycle phase parameter. MANN (Zhang et al., 2018) introduced a learned mixture of experts model, that predicts different blending weights of the motion prediction networks, based on phase parameters. Having a single global phase for the full body can lead to averaging problems for different actions for the lower and upper bodies. Local phases were introduced to address this issue (Starke et al., 2020). However, these methods require datasets labeled with phases. The work on deep phases (Starke et al., 2022) is an unsupervised approach to predicting phase parameters for an unstructured motion dataset. These phase parameters can then be used to train a MANN architecture motion controller.

Learned motion matching (Holden et al., 2020) was introduced to reduce the memory footprint of motion matching by replacing certain components with learned alternatives. They first train a pair of neural networks, called Compressor and Decompressor, to learn a mapping from a learned encoding and feature vector to the full pose. Next, they learn the Projector and Stepper networks to convert the full pose to the feature code and to step forward in time. Using the Decompressor, Projector, and Stepper networks, they can perform motion matching with large datasets without increasing time or space requirements. While achieving compression, they do not systematically remove the bias of the original.

Our approach is model and task agnostic and compatible with both architectures. It is a utility that acts on the input dataset to improve the quality of the trained models.

Another type of controller is based on a pre-trained motion Variational Auto Encoder (VAE) model (Ling et al., 2020) followed by a controller sampling points in the VAE latent space—trained using reinforcement learning. This approach is limited by the dataset bias the motion VAE was trained on and we suspect our approach would improve their results due to a more balanced dataset and sharper turns. Humor (Rempe et al., 2021) additionally learned a conditional motion prior based on the previous state.

Motion Inbetweening. A similar task tackled with deep learning is motion inbetweening (Tang et al., 2022; J Qin, 2022; Tang et al., 2023), where models are trained to generate a pose given keyframes and timing. We believe our approach could be applied to the dataset these models are trained on and could improve the performance for transitions less present in the original dataset.

Physics-Based Deep Reinforcement Learning. Another line of work uses the laws of physics to generate motions while using example motion clips to imitate in the reinforcement learning process (Bergamin et al., 2019; Yao et al., 2022; Peng et al., 2022). Using motion matching to generate movements matching target trajectories, (Bergamin et al., 2019) then makes physics-based corrections to clean the motions such as interaction with the ground. We believe this approach could be used instead of the original motion matching algorithm we use to create cleaner data for training neural controllers.

On a more global note, neural controllers have limited data to train with in terms of ground or terrain shapes, as well as actor sizes. Deep RL and physics could be an interesting way of taking a finite motion set and generating additional movements that conform to more terrains and re-targeted to different character sizes.

3 TRAJECTORY AUGMENTATION

The importance of data augmentation has been widely accepted for machine learning. However, for full body motion data, augmentation techniques beyond trivial mirroring have not been explored.

Our approach takes a finite locomotion dataset as input and creates a larger dataset by simulating the user controls over time and leveraging motion match-

ing (Clavet and Büttner, 2015) to synthesize full-body skeletal motion that conforms to the control directives. The architecture of our method can be seen in Figure 1. As user control is randomly simulated, we can increase the total dataset size by multiple folds.

3.1 Motion Matching

Motion matching can be viewed as a temporal blending between a current motion clip and the next desired motion, resulting from a nearest neighbor search.

For locomotion, the nearest neighbor search operates on a feature space comprised of current pose descriptors and samples of the immediate future motion path. The pose descriptors contain the current 3d feet positions and velocities, as well as 3d hip velocities. The motion path is made up of 2d root velocities, positions, and directions at 20, 40, and 60 frames into the future. This corresponds to the immediate future motion path projected to the ground. After searching the dataset, a clip is recovered and blended with the current motion over a time window. For a detailed description, we defer to (Clavet and Büttner, 2015) and (Holden et al., 2020).

Upon closer inspection of the LaFan1 dataset, we see that while it contained backward motions, it lacked numerous important gait and directional transitions involving backwards motion. This represents a class of trajectories that can be generated using Motion matching but lie out-of-distribution of our original dataset. Using motion matching, we introduce some of these transitions into the data.

In the following section, we describe automatically recovering these transitions without any expert cataloguing.

3.2 Generating Control Trajectories

For a robust neural locomotion controller, we need to represent a wide variety of motions. These include idling at zero velocity, walking, running, and strafing while turning at different frequencies (changing direction quickly or smoothly).

Our approach takes as input a single scale parameter w and we break down the synthesis into three parts: first, we create an overall shape of the trajectories by creating splines (Figure 2a). In a second step, we stitch them together into a long continuous path (Figure 2b). This path captures the shape of the control signals, but not the gait type, nor the velocity, such as zero velocity for idling, and higher velocities for quick changes in directions. Hence in the last part (Figure 2c), we re-parameterize the path into a uniform distribution of control signals. These include at

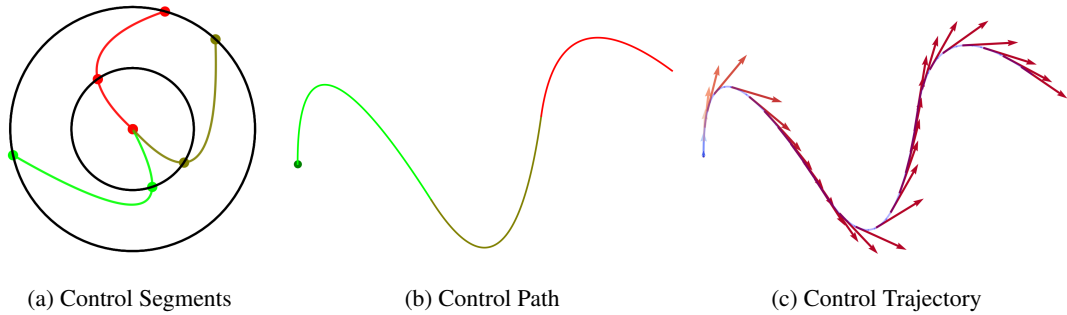


Figure 2: The Control Trajectory Generation consists of three steps. First individual smaller control segments are created randomly around the zero point seen in (a). (b) shows the resulting control path after all control segments have been stitched together and in (c) we can see the joystick signal of the extracted control trajectory that is provided to the motion matching algorithm.

times zero velocities for idling, as well as different turning frequencies (by sampling at higher speeds), together with gait parameters for idling, walking, running, and strafing. This gives us added control on proportion of each gait type and facing direction.

Control Segments Using Splines

One of the hardest tasks for a neural locomotion controller is reliably generating sharp turns. Due to poor representation in the training data and averaging artifacts, neural controllers lose the high frequency details in such motion. Wander and Seek (Ravel, 2021) are two steering algorithms that are commonly used controlling background crowd characters in games. Both algorithms modify the current velocity by an external force. The Wander algorithm uniformly samples a direction and a magnitude for the force. In contrast, the Seek algorithm adds a force in the direction of another object being “sought” with maximum magnitude. Both these algorithms perform well for background characters that are required to move predictably and not be distracting. However, the protagonist, i.e. the user controlled character, does not move similar to these background characters and needs to make sharp turns regularly. Creating sharp turns with these algorithms would require adding a large accelerations in the opposite direction to the current velocity. This requires careful calibration of the force sampling to get a good balance between moving straight, slow turns and sharp turns.

We instead use cubic splines with control points lying on concentric circles to generate the trajectories.

For each curve, we use three points $x_1, x_2, x_3 \in \mathbb{R}^2$. The starting point, x_1 , always lies at the center of two concentric circles. The second and third points are sampled uniformly on the two circles of different sizes:

$$\begin{aligned} \theta_i &\sim \mathcal{U}[0, 2\pi), \\ r_i &\sim \mathcal{U}[r_{i_{min}}, r_{i_{max}}], \\ x_2 &= \cos(\theta_1)r_1, \sin(\theta_1)r_1, \\ x_3 &= \cos(\theta_2)r_2, \sin(\theta_2)r_2 \end{aligned} \quad (1)$$

Consider the two line segments, $x_1 - x_2$ and $x_2 - x_3$. If the angle between the two line segments is large, then the cubic spline fitted to these three points will have large straight sections and a slow turn. But if the angle between the line segments is very acute, i.e. the three points are close to co-linear, then the fitted spline will produce a sharp turn close to x_2 . Since the three points are uniformly sampled, the angle between the line segments also varies randomly. Giving us a good mix of turns at different speeds.

We found the following parameters of the radii to give a good balance between the different types of turns:

$$\begin{aligned} r_{1_{min}} &= 1.0w, \\ r_{1_{max}} &= 3.0w, \\ r_{2_{min}} &= 0.2w, \\ r_{2_{max}} &= 5.0w \end{aligned} \quad (2)$$

Note in Figure 2a, how x_2 and x_3 can lie on an inner and outer circle interchangeably as the radii ranges overlap for x_2 and x_3 in Equation (2). Sampling curves as such enables us to generate a wider variety of curvatures and directions.

Control Path

We fit a cubic spline to the three points of the segment and repeat the process K times. Then stitching the segments together by integrating the last position into the next spline positions and aligning the respective end and start rotations, resulting in a long continuous control path $x(s)$ as shown in Figure 2b.

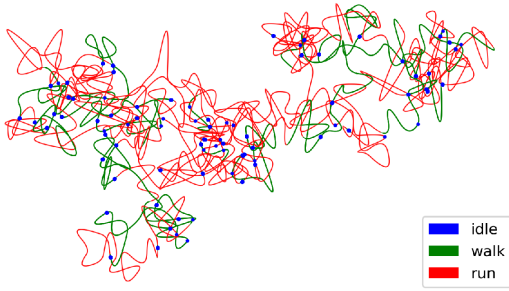


Figure 3: An example of a control trajectory holding a wide variety of both smooth and sharp turns, as well as different actions such as idling (blue), walking (green), and running (red), together with transitions between them.

Control Trajectories

Using our control path $x(s)$ that holds the shape of joystick trajectories, we create a discrete sampling of the path s_i that we call Control Trajectories, comprised of a joystick direction d_i , and action state c_i .

We start at time $s_0 = 0$ and sample a time window of length $N \sim (120, 1200)$ frames. Each frame s_i in the window will have one gait parameter that we sample from a uniform distribution $c_i \sim \mathcal{U}(\{\text{idling, walking, running}\} \times \{\text{forward, strafing}\})$ (using one-hot encoding) and we will advance from one frame to another using a velocity sampled from a uniform distribution as detailed below.

For idling, we need to emulate a control signal that is not moving, and thus set the idling velocity distribution to zero:

$$v_{idle} \sim (0, 0) \quad (3)$$

For the locomotion gaits (walking and running), we want a variety of high and low-frequency changes in directions, and thus sample uniformly between low and high values of velocities:

$$v_{locomotion} \sim (1.5w, 4.0w) \quad (4)$$

To emulate how the joystick is smoothly moving and not discretely jumping from one velocity to another, we blend the velocities over each window using a simple spring-based interpolation. Figure 2c shows the joystick direction and magnitude over the generated control path.

Using our velocity v_i , we advance in parameter space $s_{i+1} = s_i + v_i$. We then compute our control signals (joystick states) as the difference between consecutive points $d_i = x(s_{i+1}) - x(s_i)$ and bundle it together with the gait type c_i . We then use this simulated gamepad data and feed it into the motion matching algorithm to generate human motions corresponding to the control signal.

4 EXPERIMENTS

To evaluate our approach, we look at trajectory coverage (balance), foot skating, and responsiveness similar to (Lee et al., 2021). To this end we take the LaFan1 (Harvey et al., 2020) dataset and augment it with our TrajAug approach, as well as with a manually created dataset (Manual), where we record user input when controlling a gamepad. For each evaluation, we train two different neural controllers MANN (Zhang et al., 2018) and LMM (Holden et al., 2020), on the three datasets: source, Manual, and our Trajectory Augmentation (TrajAug). Note that to evaluate foot skating on a common basis, we did not utilize post-process inverse kinematics in any part of our experiments.

4.1 Source Dataset

The LaFan1 (Harvey et al., 2020) dataset contains high-quality motion clips for a number of scenarios from 5 different actors. In our implementation, we use the clips from Actor 5 relating to walking and running. The complete dataset contains about 496672 motion frames at 30fps (~ 4.6 hours). However, we use the same small subset as (Holden et al., 2020) of 14975 motion frames (~ 8.3 mins), entailing a fair comparison to their pre-trained model.

4.2 Neural Controllers

We evaluate our data augmentation using two different types of neural controllers. One is a popular phased-based neural network, the MANN architecture (Zhang et al., 2018) using DeepPhase (Starke et al., 2022) to label the data with locomotion phases. The other neural controller is from the Learned Motion Matching paper (Holden et al., 2020). The following sections provide more implementation details for these models.

4.2.1 Phase-Based Neural Controller

We implemented a phase-based neural controller using the previously mentioned models. This neural controller takes as input the current trajectory, character pose, and phases created using DeepPhase (Starke et al., 2022). Both trajectories and pose are transformed to root-local space. For the trajectory, we use positions, velocities, and orientations for 6 timesteps with stride 10 into the past and future. We use the same amount of timesteps for the phases but use a stride of 2. Additionally, our phase space has 5 channels. The mixture of experts network consists of two

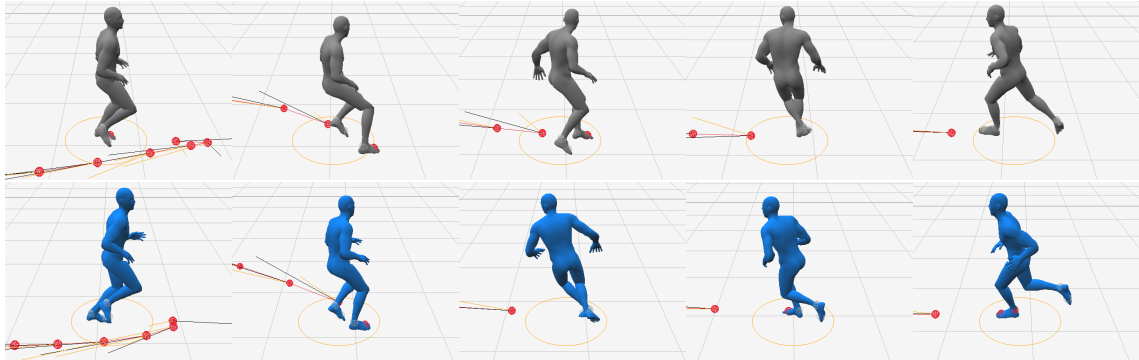


Figure 4: Here we see the difference in responsiveness between the phase-based neural controller trained on original dataset (gray) versus our augmented dataset (blue) for frames 0, 8, 16, 24 and 32. Our model is able to react faster and finish the turn before the other. The animation was recorded using the identical controller input for both models.

neural networks, the gating network and the motion prediction network as detailed in (Zhang et al., 2018). We use a size of 128 for the hidden layer in the gating network as well as 8 experts. Furthermore, the motion generation network consists of a size of 512 nodes for each hidden layer. The input is first normalized before we forward it through the network. We apply an L1 loss between the predicted output and ground truth and train with AdamWR for 250 epochs. Additionally, we use a foot skating loss with a weight of 0.1 and a bone length loss with a weight of 0.001. We train with a learning rate of 0.001 and batch size of 32 and apply a dropout of rate 0.3. We use ELU activation functions everywhere except for a Softmax in the last layer of the gating network.

4.2.2 Learned Motion Matching

The second neural controller we used for testing comes from the Learned Motion Matching (LMM) paper (Holden et al., 2020). We used their implementation directly and retrained the network on our three different datasets. For all neural networks, they use a hidden layer size of 512. The Compressor network has 5 layers while using ELU activation functions. All other networks use ReLUs where the Decompressor has 3 hidden layers, the Stepper 4, and the Projector 6 hidden layers. All networks are trained with RAdam using 500000 iterations using an initial learning rate of 0.001 that decays by 0.99 every 1000 iterations.

5 EVALUATION AND ANALYSIS

We evaluate three aspects of our approach. First, the dataset balance by measuring and visualizing the trajectory coverage, second and third are the foot skating and responsiveness using metrics defined below as well as animations shown in our video. We use two

pre-recorded control trajectories to evaluate using the metrics. Walk only entails forward motion and consists of long smooth curves as well as multiple 180° turns on the motion path in the shape of a star. The Strafe dataset includes motion forward, sideways, and backward both for walking and running with smooth and tight turns. We additionally evaluated a third test set for only forward running motion. But the results are omitted here as they were virtually the same as with the Walk dataset.

5.1 Trajectory Coverage

The first step to measure the diversity or bias of our dataset is to look at the trajectory coverage. The trajectory coverage looks at trajectory positions 30 frames into the future for every frame in the dataset. We transform the trajectories to be in root-local coordinate space such that for every trajectory the character stands at position (0,0) and the root is oriented towards the positive y-direction or up in 2d coordinates. As we can see in Figure 5 the motion capture taken from the original LaFan1 dataset has a large bias towards the front of the character. The backward direction is underrepresented. While the difference in trajectory length is due to the forward walking and running speeds being higher than the ones backward, we see a clear bias in the density of forward trajectories compared to the rest. This also affected the quality of the backward motion significantly as the neural controller showed most artifacts of foot skating in this direction when using the original dataset.

To further evaluate our approach compared to recording a user while using motion matching, we have the user-recorded Manual dataset in Figure 5 showing more trajectories with immediate backward direction. We posit that it is easier to oversee certain parts of the motion space when doing the data aug-

Table 1: Phase-based controller. Shows both foot skating and responsiveness evaluated on the phase-based neural controller with the MANN architecture and Learned motion matching. For the Phase-based controller, we see similar results in foot skating and improvements with both recorded and manually curated datasets in responsiveness on the forward walking test set. However, we achieve significant improvements of up to half for strafing. For Learned Motion Matching controller, we achieve improvements using our method across all metrics and test sets.

Test Set	Phase-based Controller				Learned motion matching			
	Foot Skating ↓		Responsiveness ↓		Foot Skating ↓		Responsiveness ↓	
	Walk	Strafe	Walk	Strafe	Walk	Strafe	Walk	Strafe
Original	0.0109	0.019	0.530	0.346	0.0051	0.0076	1.121	1.009
Manual	0.0117	0.012	0.406	0.274	0.0043	0.0055	1.091	0.909
TrajAug(ours)	0.0113	0.008	0.424	0.252	0.0039	0.0049	0.988	0.881

mentation manually. Finally, with our fully automatic TrajAug dataset we have far higher coverage in every direction.

Trajectory coverage can be seen as the variability of the trajectories across all possible angles. Quantitatively, we measure trajectory coverage by computing the standard deviation of the local trajectory density. Therefore, a low standard deviation implies uniform coverage while a large standard deviation indicates biased coverage. While the original dataset reaches a standard deviation of 0.171, the recorded dataset comes to 0.158, and our TrajAug dataset to 0.154. However, we should also note that the recorded dataset was specifically built to reduce the bias of the original dataset. Additionally, we composed our TrajAug dataset with a fully balanced dataset and a dataset biased to forward motion with standard deviations of 0.045 and 0.314 respectively. We do this to ensure good results over all possible motions but preference to forward locomotion as it represents the most important form of motion in controller applications. Hence our method allows balancing the dataset while also satisfying curation preferences based on specialist knowledge.

As we show in our video, MANN especially benefits from a more balanced dataset. It struggles to carry out any meaningful backwards motion from only the original dataset. But the models trained on both the Manual and TrajAug datasets are able to produce backwards motion.

5.2 Foot Skating

A common artifact with motion controllers is unrealistic foot skating in cases where the feet should be in fixed contact with the ground. Such foot skating often results in neural controllers due to their smooth nature which averages poses. Our foot skating metric, FS , measures how much the joints that can make contact move while in contact with the floor. We define such joints j to be in the set $J_{contact}$. However, for our purpose of locomotion, these are only the feet joints. We

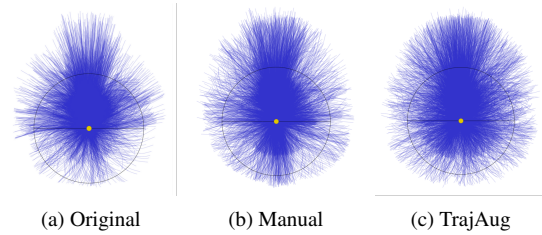


Figure 5: We randomly sampled 10000 frames into the dataset and took the 30 frames into the future for each. These are transformed to be relative to the character frame (shown as yellow circle), thus visualizing paths the character can take in the dataset. We can observe (a) the original dataset having less coverage for backward motion than (b) a manually recorded or (c) a TrajAug augmented dataset (ours). A perfect circle would represent a completely unbiased dataset.

measure the mean of the sum of velocity magnitudes $\|v\|$ for the joints that are currently in contact with the floor, as shown in Eq. (5). We decide whether or not a joint is in contact with the floor if its current height pos_z is lower than a threshold h . Different joints can have different thresholds for example the heel joint is usually placed higher than the toe. For the LaFan1 dataset, we use a height of 2.2cm for the toes and 8.7cm for the heels.

$$FS = \frac{1}{T} \sum_{t=0}^T \left(\sum_{j \in J_{contact}} \|v_j\|_2 \cdot \mathbb{I}[pos_{j,z} < h_j] \right) \quad (5)$$

Table 1 show the foot skating metrics applied to both neural controllers with each of the different datasets for Walk and Strafe motion. Especially using the Learned Motion Matching controller we see significant improvement in strafing. In the phase-based controller, we managed to reduce foot skating by more than half. Furthermore, we show improvements in walking for the Learned Motion Matching controller and similar performance with the phase-based network.

5.3 Responsiveness

For interactive applications, and in particular video games, responsiveness is very important to ensure a good user experience. To evaluate responsiveness quantitatively, we recorded controller inputs that hold multiple tight curves for both walking and strafing. For each curve in the predefined set C , we measure the responsiveness, R , which corresponds to the delay between the apex of the joystick T_i^J and the apex of the resulting motion path T_i^M as defined in:

$$R = \frac{1}{|C|} \sum_{i=0}^{|C|} (T_i^M - T_i^J), \forall i \in C, \quad (6)$$

Qualitatively, Figure 4 shows how the neural controller trained with our augmented dataset has already completed its turn about 8 frames early while the neural controller trained with the original dataset is still in the middle of the rotation.

Evaluating this metric for the phase-based MANN architecture on our three datasets in Table 1, we can see a drop in delay from the original dataset for both augmented datasets. Interestingly, we see a slight improvement in the recorded dataset over ours for walking with 180° turns in Table 1. After the fact, we observed that our recorded dataset had many such turns using walking and strafing.

For the Learned Motion Matching neural controller, we observe improvement overall shown in Table 1. We think this could be due to the blending occurring in Learned Motion Matching, resulting in less benefit to responsiveness. For foot skating, we have seen a high improvement as our data augmentation balanced the dataset and reduced the number of extrapolation errors.

As the original data already well represents walking forwards, we see less improvements in Walk motion across the models and the metrics. In contrast, the originally worse represented Strafe motion benefits significantly for both models. This further highlights the need to balance the datasets used for training neural controllers.

6 CONCLUSION

We proposed a new trajectory driven data augmentation technique for Motion Capture data reducing foot skating and pose blocking artifacts in Neural Locomotion Controllers. The resulting dataset further includes sharper turns and removes undesirable biases compared to the original mocap making it more suitable for in-game locomotion controllers. We evaluated our approach on trajectory coverage, foot skat-

ing, and responsiveness and showed that we obtain superior results compared to training on only the original dataset.

In this work, we focused on locomotion including idling, walking, running, and strafing, and we leave experimenting with other actions and interactions with the scene as future work. Similarly, we leave experimenting with more random trajectory generating solutions as future work as this is not the main novelty of our work. Using our framework to extrapolate further away from the original dataset may require replacing motion matching with a more robust frame sampling method such as (Bergamin et al., 2019). If different styles are labeled in the data, motion matching can be easily modified to produce stylized locomotion. We have also witnessed in recent years motion matching extended for character-scene interactions in games such as *Fifa* and *Madden* (Allen, 2021). Using this interaction framework could help extend this data augmentation technique beyond locomotion and include more complex motion and interactions.

REFERENCES

- Allen, H. (2021). Animation summit: Environmental and motion matched interactions; 'madden', 'fifa' and beyond!
- Bergamin, K., Clavet, S., Holden, D., and Forbes, J. R. (2019). Drecon: data-driven responsive control of physics-based characters. *ACM Transactions On Graphics (TOG)*, 38(6):1–11.
- Chin-Cheong, K., Sutter, T., and Vogt, J. E. (2019). Generation of heterogeneous synthetic electronic health records using gans. In *workshop on machine learning for health (ML4H) at the 33rd conference on neural information processing systems (NeurIPS 2019)*. ETH Zurich, Institute for Machine Learning.
- Clavet, S. and Büttner, M. (2015). Motion matching - the road to next gen animation. https://www.youtube.com/watch?v=z_wpgHFSWss&t=658s.
- Harrower, G. (2018). Real player motion tech in 'EA sports UFC 3'.
- Harvey, F. G., Yurick, M., Nowrouzehzahr, D., and Pal, C. (2020). Robust motion in-betweening. 39(4).
- Henter, G. E., Alexanderson, S., and Beskow, J. (2020). Moglow: Probabilistic and controllable motion synthesis using normalising flows. *ACM Transactions on Graphics (TOG)*, 39(6):1–14.
- Holden, D., Kanoun, O., Perepichka, M., and Popa, T. (2020). Learned motion matching. *ACM Transactions on Graphics (TOG)*, 39(4):53–1.
- Holden, D., Komura, T., and Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):1–13.

- J Qin, Y Zheng, K. Z. (2022). Motion in-betweening via two-stage transformers. *ACM Transactions on Graphics (TOG)*.
- Lee, K., Lee, S., and Lee, J. (2018). Interactive character animation by learning multi-objective control. *ACM Trans. Graph.*, 37(6).
- Lee, K., Min, S., Lee, S., and Lee, J. (2021). Learning time-critical responses for interactive character control. *ACM Trans. Graph.*, 40(4).
- Li, B., Hou, Y., and Che, W. (2022). Data augmentation approaches in natural language processing: A survey. *Ai Open*, 3:71–90.
- Ling, H. Y., Zinno, F., Cheng, G., and Van De Panne, M. (2020). Character controllers using motion vaes. *ACM Transactions on Graphics (TOG)*, 39(4):40–1.
- Peng, X. B., Guo, Y., Halper, L., Levine, S., and Fidler, S. (2022). Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.*, 41(4).
- Ravel, K. (2021). Steering behaviors. <https://www.kaspar.wtf/blog/steering-behaviors>.
- Rempe, D., Birdal, T., Hertzmann, A., Yang, J., Sridhar, S., and Guibas, L. J. (2021). Humor: 3d human motion model for robust pose estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11488–11499.
- Starke, S., Mason, I., and Komura, T. (2022). Deep-phase: periodic autoencoders for learning motion phase manifolds. *ACM Transactions on Graphics (TOG)*, 41(4):1–13.
- Starke, S., Zhang, H., Komura, T., and Saito, J. (2019). Neural state machine for character-scene interactions. *ACM Trans. Graph.*, 38(6):209–1.
- Starke, S., Zhao, Y., Komura, T., and Zaman, K. (2020). Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics (TOG)*, 39(4):54–1.
- Tang, X., Wang, H., Hu, B., Gong, X., Yi, R., Kou, Q., and Jin, X. (2022). Real-time controllable motion transition for characters. *SIGGRAPH 2022*.
- Tang, X., Wu, L., Wang, H., Hu, B., Gong, X., Liao, Y., Li, S., Kou, Q., and Jin, X. (2023). Rsmt: Real-time stylized motion transition for characters. *SIGGRAPH 2023*.
- Yao, H., Song, Z., Chen, B., and Liu, L. (2022). Controlvae: Model-based learning of generative controllers for physics-based characters. *ACM Transactions on Graphics (TOG)*, 41(6):1–16.
- Zhang, H., Starke, S., Komura, T., and Saito, J. (2018). Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics (TOG)*, 37(4):1–11.