Local Motion Phases for Learning Multi-Contact Character Movements

SEBASTIAN STARKE, University of Edinburgh, UK and Electronic Arts, USA
YIWEI ZHAO, Electronic Arts, USA
TAKU KOMURA, University of Edinburgh, UK
KAZI ZAMAN, Electronic Arts, USA

Fig. 1. A selection of results using our method to generate ball-dribbling movements and interaction behaviours with other characters.

Training a bipedal character to play basketball and interact with objects, or a quadruped character to move in various locomotion modes, are difficult tasks due to the fast and complex contacts happening during the motion. In this paper, we propose a novel framework to learn fast and dynamic character interactions that involve multiple contacts between the body and an object, another character and the environment, from a rich, unstructured motion capture database. We use one-on-one basketball play and character interactions with the environment as examples. To achieve this task, we propose a novel feature called local motion phase, that can help neural networks to learn asynchronous movements of each bone and its interaction with external objects such as a ball or an environment. We also propose a novel generative scheme to reproduce a wide variation of movements from abstract control signals given by a gamepad, which can be useful for changing the style of the motion under the same context. Our scheme is useful for animating contact-rich, complex interactions for real-time applications such as computer games.

CCS Concepts: • **Computing methodologies** → **Motion capture**; *Neural networks.*

Additional Key Words and Phrases: neural networks, human motion, character animation, character control, character interactions, deep learning

Authors' addresses: Sebastian Starke, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, UK, sebastian.starke@ed.ac.uk, Electronic Arts, USA, sstarke@ea.com; Yiwei Zhao, yiwzhao@ea.com, Electronic Arts, USA; Taku Komura, tkomura@ed.ac.uk, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, UK; Kazi Zaman, kzaman@ea.com, Electronic Arts, USA.

0730-0301/2020/7-ART1 $15.00
https://doi.org/10.1145/3386569.3392450

ACM Reference Format:
Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Trans. Graph.* 39, 4, Article 1 (July 2020), 14 pages. https://doi.org/10.1145/3386569.3392450

1 INTRODUCTION

There is a huge demand in simulating fast and complex interactions that involve multiple contacts between a character and objects, an environment, and other characters, especially in computer games and films. For example, for basketball games, the players need to dribble the ball while making various movements with different foot-fall patterns to compete with the opponent characters.

Previous techniques to learn from unstructured motion capture database have limitations in terms of scalability, realism and variation in the movements. Firstly, most techniques require aligning the motions by a global temporal parameter such as the phase, which is often difficult when the motion involves multiple contacts that are asynchronous. Secondly, even when the motions are learned by the controller, there can be issues reproducing a wide variation of movements from low dimensional control signals such as those provided by the user through keyboards or gamepads.

In this paper, we propose a novel data-driven framework to learn fast and dynamic interactions that involve multiple contacts. We make use of a large database of one-on-one basketball play as our main example, where one player catches, dribbles and plays tricks with the ball, while avoiding the opponent player who tries to defend and intercept the ball. We design and train a neural character controller that can learn and produce realistic offense and defense actions under a unified framework, so that the players can easily switch from the offense to the defense during the play.

To let the model learn movements that involve fast and complex interactions where the contacts between the body and the ball or the ground quickly switch in an asynchronous manner, we propose

a feature that we call *local motion phase*, which is defined based on how each body part contacts external objects. The feature can be computed automatically from unstructured motion capture data using an evolutionary strategy. Using the local motion phase, the network can learn the motion of individual body parts locally without aligning the entire body motion using a global phase, which is often difficult for fast and complex interactions that happens during basketball plays.

To cope with the ambiguity between the low dimensional control signal and a rich set of full body motion, we propose a novel generative model that can reproduce a wide variation of sharp movements conditioned on the high-level control signal. Our generative control model can convert an abstract control signal produced from the user instructions into a wide variation of sharp control signals that can be mapped to realistic full body motion with subtleties.

Once the system is trained from a large amount of motion capture data of basketball plays, the user can interactively control the character to produce fast and asynchronous basketball skills, such as dribbling, feinting, stealing and defending in real-time, which can be useful for computer games and VR sports training. Our system is also useful for learning other contact-rich movements, such as sitting on a chair, opening a door and quadruped locomotion, in higher quality compared to previous models, without any human labelling.

The contributions of this paper can be summarized as follows:

- a neural character controller that can synthesize a large variety of dynamic, asynchronous movements, such as movements in basketball, in high quality,
- a scheme to automatically extract phase variables at local level that can robustly align the motion sequences (see Section 5),
- a generative model that can convert an abstract, high-level user control signal into a wide variation of sharp signals that can be mapped to realistic character movements with subtleties (see Section 6) and
- an evaluation of the scheme in comparison to existing approaches (see Section 9).

2 RELATED WORK

In this section, we review data-driven animation techniques that are applicable for animating fast and dynamic movements with multiple contacts. These can be classified into (1) motion blending techniques that explicitly classify, segment and align motions, (2) time-series models that do not require explicit motion alignment and (3) physically-based methods that refer to/learn controllers from motion capture data.

Learning Motion by Temporal Alignment. For synthesizing novel motion using the motion capture data, a straightforward approach is to align the motions of the same class along the timeline and blend them with weights computed by the controller [Kovar and Gleicher 2004; Min and Chai 2012; Rose et al. 1998; Rose III et al. 2001; Wiley and Hahn 1997]. Aligning the motion along the timeline are either done manually/semi-automatically [Shin and Oh 2006], by dynamic time warping (DTW) [Kovar and Gleicher 2004; Mukai and Kuriyama 2005] or using contact states [Min and Chai 2012; Safonova and Hodgins 2007]. Shin and Oh [2006] introduce the idea of fat graphs, an extension of the motion graphs structure [Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002] where the edge represents a set of motions that can be interpolated to synthesize a novel motion. Heck and Gleicher [2007] construct a similar data structure where the motions to be interpolated are aligned based on nearest neighbor search and DTW [Kovar and Gleicher 2004].

Temporally aligning the motions based on the contact helps to avoid effects such as foot skating. Safonova and Hodgins [2007] construct interpolated motion graphs and only interpolate motions that start/end with the same contact states. Min et al. [2012] propose a similar data structure called motion graph++, where the motions are represented by functional PCA; the optimal series of motions that satisfy the constraints are computed by maximum a posteriori estimation. Zhao et al. [2013] enhance the approach to synthesize physically-plausible grasping behavior.

For motions such as those in basketball, the foot contact and hand-ball contacts can happen asynchronously and switch very quickly. Thus, explicitly classifying and aligning a large database of motions based on contacts is not feasible.

Time Series Models. Time series models are those where the current pose of the character is predicted from the previous motion and possibly a control signal. Thanks to its nature, such models can be used for real-time character control, which is our target application. Human motion has been modelled with various time series models, such as conditional Restricted Bolzmann Machine (cRBM) [Taylor et al. 2007], Gaussian processes (GP) [Wang et al. 2008], recurrent neural networks [Fragkiadaki et al. 2015; Harvey and Pal 2018; Lee et al. 2018; Li et al. 2017; Villegas et al. 2018], Phase Functioned Neural Networks (PFNN) [Holden et al. 2017] and mixture-of-experts models [Starke et al. 2019; Xia et al. 2015; Zhang et al. 2018].

LSTM-based approaches have been applied for motion prediction [Fragkiadaki et al. 2015; Li et al. 2017], motion retargeting [Villegas et al. 2018], keyframe animation [Harvey and Pal 2018] and interactive character control [Lee et al. 2018]. The difficulty of using LSTM is in the tuning of its meta parameters: simple models cannot produce realistic motions but overly complex models do not generalize well. Especially for interactive character control, when the internal memory state is high dimensional, they often suffer from low responsiveness due to the large variation of the memory state [Starke et al. 2019]. Given the user instruction, the system needs to keep updating the memory until it reaches a state observed during training for the character to start follow the user instructions. Lee et al. [2018] overcome this by conducting a significant amount of data augmentation; they expand 10-12min of data into 4 hours by motion editing. On the other hand, this results in less variation of the motion during runtime as the original motion dataset is not very large. We tackle a problem of using a large motion capture dataset to produce a wide variation of the motion during runtime.

Providing the phase variable [Holden et al. 2017], which represents the progression of the motion, helps to improve the quality of the motion computed by time-series models. Holden et al. [2017] define the phase based on the foot contact of bipedal locomotion and produce hiqh quality locomotion that can adapt to different terrain geometry. On the other hand, defining a phase for movements that involve complex contact patterns requires handcrafting rules

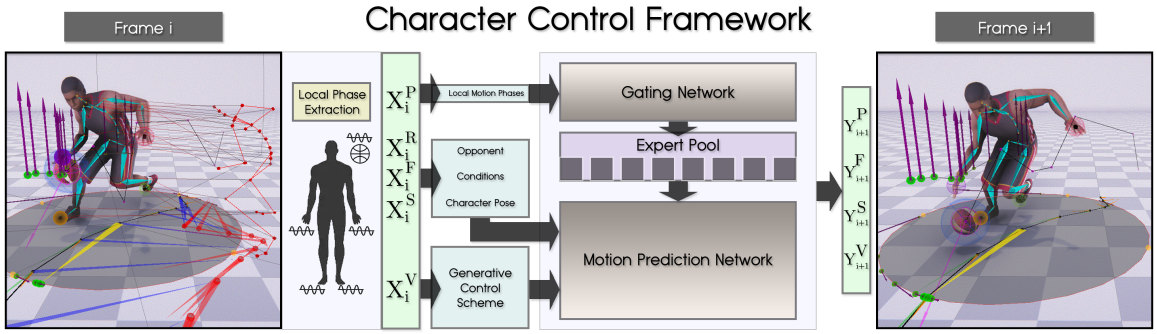

Fig. 2. The architecture of our system composed of the gating network and the motion prediction network. The gating network takes as input the local phase segments, and computes the expert blending coefficients which are then used to generate the motion prediction network. The motion prediction network takes as input the posture and user control variables to predict the motion from one frame into the next.

for defining the phase [Starke et al. 2019]. This is not feasible for movements in basketball, where there is a wide range of fast and dynamic movements.

Physically-based Character Animation. Physically-based character animation, where the characters are controlled kinematically using physical rules as constraints, or controlled by torques under physical environments, can be applied for synthesizing motion that involves fast, dynamic contacts.

Methods such as spacetime constraints [Witkin and Kass 1988] let users provide the contact pattern as conditions, and then optimize the motion using physically-based constraints. Liu and Popović [2002] compute humanoid jumping motions by spacetime constraints. As specifying such contact patterns can be difficult for fast and dynamic movements, Ye and Liu [2012] propose to predict such contacts by evolutionary strategies. The focus of these methods is to apply optimization to compute a motion that satisfies physical constraints. In our research, we are more interested in learning the way humans produce such contacts as part of learning complex behaviors.

Another direction of physically-based character control is the forward dynamics approach [Coros et al. 2010; Hodgins et al. 1995; Raibert and Hodgins 1991; Yin et al. 2007], where joint torques are computed and applied to the body to synthesize realistic movements. Simulating movements with multiple contacts is known to be a difficult problem in such frameworks due to the instability introduced by the contacts. Liu et al. [2010] propose a method to randomly add minor offsets to the motion to guide the body to follow a given motion capture trajectory. Liu et al. [2016] learn a model with control graph and linear feedback policies that can produce character movements according to the user inputs in a stable manner. This idea is further extended to learn basketball dribbling [Liu and Hodgins 2018], where deep reinforcement learning is applied to learn the complex arm motion to control the ball. The amount of skills that can be learned by this framework is limited, and there is a scalability issue for learning a wide range of motions from a large motion capture database due to the intense computation for openloop and feedback policies.

DeepMimic [Peng et al. 2018] is proposed as a general deep reinforcement learning framework to follow motion capture data in a physically-based environment. As it is designed to only follow a short motion clip, Park et al. [2019] and Bergamin et al. [2019] propose frameworks to follow longer sequences of motions produced by a small motion graph [Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002] or motion matching [Clavet 2016]. The scalability of these methods is yet to be explored as they are trained with a relatively small dataset. Our technique can potentially replace the motion matching module of [Bergamin et al. 2019] for faster synthesis of reference motions.

A different option to let a character follow a reference motion capture data in a physically-based environment is model predictive control: Hong et al. [2019] create a soccer player controller to dribble a ball, and show its robustness with respect to other players tackling. iLQG [Todorov and Li 2005] is used to compute the optimal set of torques and gains to follow a reference motion. The reference motion is selected by nearest neighbor search: again, our proposed method can potentially be applied as a motion planner for this approach.

In summary, learning a rich set of interactions where the contacts quickly switch in a complex manner from a large motion capture dataset without manual human labelling is a problem yet to be solved, either in a the kinematics or physically-based domain. We propose a novel scheme to learn such complex skills from a very large motion capture database.

3 SYSTEM OVERVIEW

Our deep learning framework is a mixture-of-experts scheme similar to [Zhang et al. 2018] and [Starke et al. 2019]. The system consists of the motion prediction network and the gating network (see Fig. 2). The gating network computes a set of expert weights, and learns how to dynamically combine them via blending coefficients to construct the motion prediction network. The motion from one frame into the next is then computed in an autoregressive fashion from the current character state and the user-given control commands.

To enable the framework to learn basketball movements like quick ball maneuvers or player-interaction movements from a large

motion database, we propose two main enhancements: first, training the system with local motion phases, and second, a generative control model that takes as input the raw high-level user control commands and generates a sharper variety of control signals.

The local motion phases are computed individually for each bone that makes contacts with an object/environment and can encode asynchronous movements of the limbs. This is fundamentally different to the work in [Starke et al. 2019], where movements of different actions are each assumed to be synchronised by a single global phase variable - this requires a careful labeling process or defining explicit rules about when the action is starting and ending, which can be difficult and also sometimes ambiguous. The local motion phase is computed by a uniform rule based on contacts of the individual bones, and is simple to compute automatically.

The generative control model is introduced to produce a wide variation of sharp movements from coarse high level control signals provided by the user. When there are many motions that correspond to the same input signal, a simple regression by a deterministic model will result in an averaged motion where the sharpness is lost with little variation. We cope with this problem by introducing a generative model that takes as input the raw high-level user control commands and random noise, and generates a sharper sequence of control signals from that. This enables the system to not only produce motions in higher quality, but also to generate variations in a non-deterministic fashion.

In the following, we will first describe about the inputs and outputs of our system in Section 4. We then introduce the local motion phase in Section 5, followed by the generative control model in Section 6. The network training process is described in Section 7 and the user interface for character control is described in Section 8. The experiments and evaluation are given in Section 9.

4 SYSTEM INPUTS AND OUTPUTS

Our system is a time-series model that predicts the state variables of the character, the ball etc. in the next frame $i+1$ given those in the current frame i. The inputs and outputs are designed such that our system can produce close interactions between the character and an object, an environment and another character. Some variables for control and conditioning are application oriented: here we mainly describe in the basketball setup, although the concept is general and applicable to other motions such as maneuvers of objects and interactions with the environment. For training, the Cartesian features in the input and output are transformed into the root coordinate system of the character at frame i and $i+1$, respectively. All features live in a time series window [1] $\mathcal{T}_{-1s}^{1s}$ within which data of 13 uniformly-sampled points (6 each in the future and past 1s window, and one for the current frame) are collected. How the values are extracted from the motion capture data is explained in Appendix A.1.

Inputs. The complete input vector $\mathbf{X}_i$ at frame i consists of five components $\mathbf{X}_i = \{\mathbf{X}_i^S, \mathbf{X}_i^{\mathcal{V}}, \mathbf{X}_i^{\mathcal{F}}, \mathbf{X}_i^{\mathcal{R}}, \mathbf{X}_i^{\mathcal{P}}\}$ where each item is described below.

- **Character State** $\mathbf{X}_i^S = \{\mathbf{p}_i, \mathbf{r}_i, \mathbf{v}_i\}$ represents the state of our character with $\mathbb{B} = 26$ bones at the current frame i. it consists of the bone positions $\mathbf{p}_i \in \mathbb{R}^{3B}$, bone rotations $\mathbf{r}_i \in \mathbb{R}^{6B}$ and bone velocities $\mathbf{v}_i \in \mathbb{R}^{3B}$, where each bone rotation is formulated by its pair of Cartesian forward and up vectors to create an unambiguous and continuous interpolation space [Zhang et al. 2018].
- **Control Variables** $\mathbf{X}_i^{\mathcal{V}} = \{\mathbf{T}_i^p, \mathbf{T}_i^r, \mathbf{T}_i^v, \mathbf{I}_i^p, \mathbf{I}_i^m, \mathbf{A}_i\}$ are the variables used to guide the character to conduct various basketball movements. It consists of the following channels that are sampled in the past-to-current time window $\mathcal{T}_{-1s}^{1s} = 13$.
  - *Root Trajectory* $\mathbf{T}$*:* For controlling the character locomotion, we train our system on the horizontal path of trajectory positions $\mathbf{T}_i^p \in \mathbb{R}^{2\mathcal{T}}$, trajectory directions $\mathbf{T}_i^r \in \mathbb{R}^{2\mathcal{T}}$ and trajectory velocities $\mathbf{T}_i^v \in \mathbb{R}^{2\mathcal{T}}$ (see Fig. 3, top left).
  - *Interaction Vectors* $\mathbf{I}$*:* A set of 3D pivot vectors $\mathbf{I}_i^p \in \mathbb{R}^{3\mathcal{T}}$ and its derivative $\mathbf{I}_i^m \in \mathbb{R}^{3\mathcal{T}}$ around the character, that together define the dribbling direction, height and speed to direct a wide range of dynamic ball interaction movements and maneuvers (see Fig. 3, top right). We describe further details about interaction vectors in Appendix A.1.
  - *Action Variables* $\mathbf{A}$*:* The action variables $\mathbf{A}_i \in \mathbb{R}^{4\mathcal{T}}$ consist of four actions that are defined as $\mathbf{A} = \{Idle, Move, Control, Hold\}$, where each of them is between 0 and 1.

  During runtime, the interaction vectors produce different effects according to the action variables and the state of the opponent. If the action state is in *Move* and *Dribble*, the character will dribble the ball, and if in *Stand* and *Hold*, the character will move the ball to the target location. When using the *Shoot* action, the interaction vectors control height and speed for throwing the ball. If the character is not controlling or holding the ball, $\mathbf{I}_i^P$ and $\mathbf{I}_i^M$ are controlled by the opponent character, and induce the user character to produce defence motion.
- **Conditioning Features** $\mathbf{X}_i^{\mathcal{F}} = \{\hat{\mathbf{B}}_i^p, \hat{\mathbf{B}}_i^v, \mathbf{B}_i^w, \mathbf{C}_i\}$ are composed of the following items that are each sampled along the past-to-current time series window $\mathcal{T}_{-1s}^{0s} = 7$.
  - *Ball Movement* $\hat{\mathbf{B}}$*:* We use the past movement of the ball of positions $\hat{\mathbf{B}}_i^p \in \mathbb{R}^{3\mathcal{T}}$ and velocities $\hat{\mathbf{B}}_i^v \in \mathbb{R}^{3\mathcal{T}}$ to guide the prediction for next frame. This conditioning is helpful since ball movement is typically highly fast-paced. We further give the ball control weights $\mathbf{B}_i^w \in \mathbb{R}^{\mathcal{T}}$ as input to the network, which were used to transform the original ball parameters from $\mathbf{B}_i$ to $\hat{\mathbf{B}}_i$ in order to learn the ball movement only within a control radius around the character (see Appendix A.2).
  - *Contact Information* $\mathbf{C}$*:* Similarly, we condition the generated motion on the contacts $\mathbf{C}_i \in \mathbb{R}^{5\mathcal{T}}$ for feet, hands and ball that appeared during the past to stabilize the movements (see Fig. 3, left bottom).
- **Opponent Information** $\mathbf{X}_i^{\mathcal{R}} = \{\mathbf{w}_i, \mathbf{d}_i, \mathbf{g}_i^p, \mathbf{g}_i^r, \mathbf{g}_i^v\}$ are the variables that describe the state of the opponent character with respect to the user character. Those consist of $\mathbf{w}_i \in \mathbb{R}^{\mathcal{T}}$, which are labels that tell if the opponent is within the 5 meter radius (1 if within the radius, otherwise 0), and $\mathbf{g}_i^p, \mathbf{g}_i^r, \mathbf{g}_i^v \in \mathbb{R}^{2\mathcal{T}}$ are 2D vectors between position samples of the user and opponent trajectories, as well the direction and velocity of the opponent trajectory, all in the time

[1] We use the notation $\mathcal{T}_{t_0}^{t_1} = N$ to describe that we collect N samples of data within a time window of $t_0 \leq t \leq t_1$.

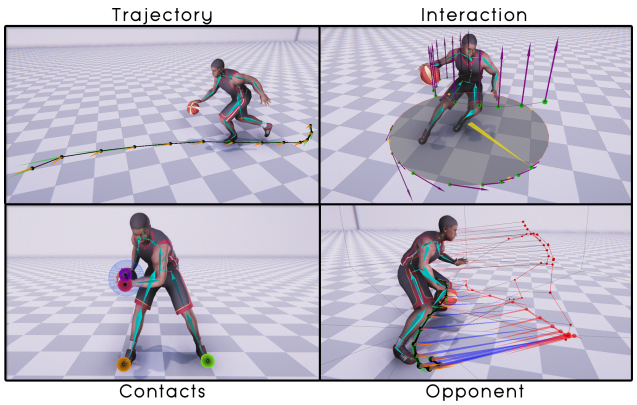

Fig. 3. Some of the input features used to predict the character pose in the next frame: (from top left to right bottom) The root trajectory $\mathbf{T}$, interaction vectors $\mathbf{I}$, contact information $\mathbf{C}$ and opponent information $\mathbf{X}^{\mathcal{R}}$.

window of $\mathcal{T}^{1s}_{-1s} = 13$. Those are further weighted using w_i which makes sure they are only active when the opponent is close, and are also required to handle captured data where no opponent is available. Finally, $\mathbf{d}_i \in \mathbb{R}^B$ are the distance pairs within $5m$ radius between the corresponding $B = 26$ joints of the two characters at current frame i (see Fig. 3, right bottom).

- **Local Motion Phases** $\mathbf{X}^{\mathcal{P}}_i = \Theta_i \in \mathbb{R}^{2K\mathcal{T}}$ are each represented by 2D phase vectors of changing amplitude for $K = 5$ key bones for feet, hands and ball, and are sampled along the past to future time series window $\mathcal{T}^{1s}_{-1s} = 13$. The details of the bone-level phase are described in Section 5.

The bone-level phases $\mathbf{X}^{\mathcal{P}}_i$ are fed into the gating network, the control variables $\mathbf{X}^{\mathcal{V}}_i$ are fed into the generative controller, and the rest are fed into the motion prediction network.

Outputs. The output vector $\mathbf{Y}_{i+1} = \{\mathbf{Y}^{S}_{i+1}\mathbf{Y}^{\mathcal{V}}_{i+1}, \mathbf{Y}^{\mathcal{F}}_{i+1}, \mathbf{Y}^{\mathcal{P}}_{i+1}\}$ for the next frame $i + 1$ is computed by the motion prediction network, and consists of the following four components.

- **Character State** $\mathbf{Y}^{S}_{i+1} = \{\mathbf{p}_{i+1}, \mathbf{r}_{i+1}, \mathbf{v}_{i+1}\}$ is pose and velocity of the character at next frame $i + 1$ with $B = 26$ bones.
- **Future Control Variables** $\mathbf{Y}^{\mathcal{V}}_{i+1} = \{\mathbf{T}^{p}_{i+1}, \mathbf{T}^{r}_{i+1}, \mathbf{T}^{v}_{i+1}, \mathbf{I}^{p}_{i+1}, \mathbf{I}^{m}_{i+1}, \mathbf{A}_{i+1}\}$ are the future control signals each sampled along the current to future time series window $\mathcal{T}^{1s}_{0} = 7$ of the next frame $i + 1$. During runtime, these features are blended with the given user-guided control signals and fed into the input in the next frame, such that the character produces plausible motion while following the user instruction:

$$\mathbf{X}^{\mathcal{V}}_{i+1} = (1 - t^{\tau})\mathbf{X}^{\mathcal{V}}_{\text{user}} + t^{\tau}\mathbf{Y}^{\mathcal{V}}_{i+1}, \tag{1}$$

where $\mathbf{X}^{\mathcal{V}}_{\text{user}}$ are the control signals produced from the user inputs by hand-crafted rules, t ranges from 0 to 1 as the trajectory gets further into the future, and τ represents an additional bias that controls the responsiveness of the character.

- **Conditioning Features** $\mathbf{Y}^{\mathcal{F}}_{i+1} = \{\hat{\mathbf{B}}^{p}_{i+1}, \hat{\mathbf{B}}^{r}_{i+1}, \hat{\mathbf{B}}^{v}_{i+1}, \mathbf{B}^{w}_{i+1}, \mathbf{C}_{i+1}\}$ are computed for the next frame $i + 1$. Note that the output also includes the delta ball rotation $\hat{\mathbf{B}}^{r}_{i+1}$, which is used to update the orientation of the ball in the next frame. The predicted weights $\hat{\mathbf{B}}^{v}_{i+1}$ are used again to transform the ball coordinates to the real world values (see Appendix A.2).
- **Local Motion Phase Updates** $\mathbf{X}^{\mathcal{P}}_{i+1} = \{\Theta_{i+1}, \Delta\Theta_{i+1}\}$ are computed in a fully autoregressive fashion, and contain the phase vectors $\Theta_{i+1} \in \mathbb{R}^{2K\mathcal{T}}$ as well as their updates $\Delta\Theta_{i+1} \in \mathbb{R}^{2K\mathcal{T}}$ for the $K = 5$ key bones, covering the current to future time series samples $\mathcal{T}^{1s}_{0s} = 7$ of the next frame. Since the network can update all phases in an asynchronous and independent fashion and in order to prevent error accumulation over multiple frames, we perform an interpolation to compute the new updated phase vectors Θ'_{i+1}, which keeps their combination in a well-defined manifold:

$$\Theta'_{i+1} = \lambda\Theta_{i+1} + (1 - \lambda)(\Theta_i + \Delta\Theta_{i+1}). \tag{2}$$

5 LOCAL MOTION PHASE

In this section, we describe our novel feature that we call the local motion phase, which boosts the system to learn movements where different parts of the body move asynchronously, such as those during basketball plays. We first describe the motivation of introducing the local motion phase, and then how to compute them from the motion capture data.

5.1 Motivation of Using the Local Motion Phase

The use of a phase variable is typically defined based on semantically meaningful start and end poses e.g. according to the foot-contact pattern, and introduces a strong connection between timing and movement. Such alignment is particularly helpful for neural networks to generate high-quality human locomotion [Holden et al. 2017] and scene interaction movements [Starke et al. 2019]. Intuitively, there are two main reasons why such phase information leads to improved quality: First, it clusters the motion in a way that the network only requires learning a smaller subset of poses, actions and transitions for each phase state. Second, it forces the animation to keep moving forward in time, whereas autoregressive motion generators can otherwise easily get stuck in poses of similar timing, resulting in unresponsive moments or missing motion details.

However, defining a global phase variable for asynchronous movements where different body parts move at different and consistently changing frequencies/phase shifts, such as when playing basketball, is extremely difficult or sometimes strictly not possible. In turn, any motion that does not follow an identical pattern would inevitably become blended with different motion states, which makes a single global phase difficult to scale and impractical to be applied to unstructured motion data.

To cope with this issue, we introduce the concept of local motion phase, which inverts the original concept from using a single global phase to instead describing the character motion by a set of multiple independent and local phases for each bone. Here we define the phase based on a simple rule; the contact transitions between the bone and other objects/environment. This allows us to extract the phase in a fully-automatic fashion (see Section 5.2), and also make it generic to arbitrary movements. Using the local phase, the system can learn to align the local limb movements individually, while also integrating their movements to produce realistic full-body behavior.

5.2 Computing the Local Motion Phase from Motion Capture Data

The local phase is computed as a preprocess from the motion capture data, by fitting a sinusoidal function

$$\Omega(\mathbf{F}_i, t) = a_i \cdot \sin(f_i \cdot t - s_i) + b_i, \tag{3}$$

parameterized by $\mathbf{F}_i = \{a_i, f_i, s_i, b_i\}$ (i is the frame index), to a filtered block function $\mathbf{G}(t)$ that represents the contact between the bone and the object/environment (see Fig. 4).

This fitting is done by inferring $\mathbf{F}_i$ at every frame i that minimizes the following RMSE loss defined in a window of N frames centered at frame i:

$$\mathcal{L}(\mathbf{F}_i) = \sqrt{\frac{\sum_t (\Omega(\mathbf{F}_i, t) - \mathbf{G}(t))^2}{N}}. \tag{4}$$

$\mathbf{G}(t)$ is computed by first normalizing the original block function (value set to 0 if no contact, and 1 if in contact)[2] in a window W of 60 frames (1 second) centered at frame j:

$$\mathbf{G}_j = \frac{c^j - \mu_{\mathbf{C}^j_W}}{\sigma_{\mathbf{C}^j_W}} \tag{5}$$

where c^j is the original block function value, and $\mu_{\mathbf{C}^j_W}$ and $\sigma_{\mathbf{C}^j_W}$ are the mean and standard deviation of contacts within that window, and then applying a Butterworth low-pass filter to the entire domain:

$$\mathbf{G}(t) = \mathcal{B}(\mathbf{G}). \tag{6}$$

After applying this normalization, shorter contacts result in larger positive values that are surrounded by smaller negative values and vice versa to maintain similar importance for different contact duration (see Fig. $4_{(1,2,3)}$). The window size N in Eq. (4) is 60 (1 second) but is adjusted according to the frequency of the surrounding contacts. The cut-off frequency of the Butterworth low-pass filter is set to 3.25, which is computed based on the Shannon-Nyquist sampling theorem with respect to the time series window of $\mathcal{T}^{1s}_{-1s} = 13$ samples that are trained in the network.

After optimizing Eq. (4), a 1D phase value can be computed by

$$\phi_i = (f_i \cdot t - s_i) \bmod 2\pi \in \mathbb{R}^1 \tag{7}$$

[2]Note that the contact labels are extracted automatically; see Appendix A.3 for the process.

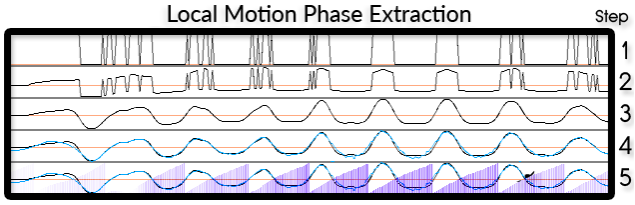

Fig. 4. Phase extraction method example applied to a single bone. The raw contact information in (1) is normalized in (2) and low-pass filtered in (3). The cyan curve in (4) then shows the fitting reconstruction, and the extracted phase values are visualized in blue in (5). The height of the purple bars illustrates the phase, the opacity illustrates the amplitude, and the slope of the successive bars illustrates the frequency.

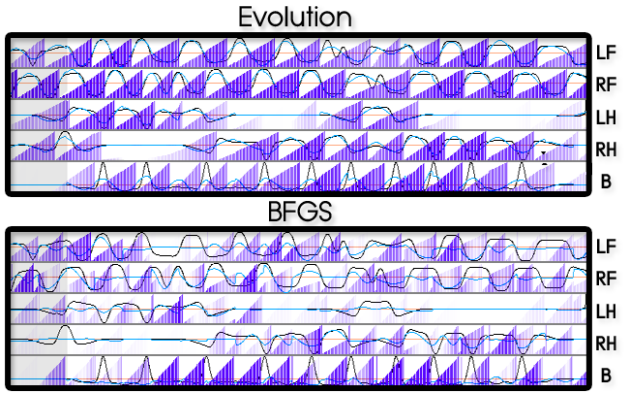

Fig. 5. Examples of local motion phase extractions for feet (LF,RF), hands (LH,RH) and ball (B). The clip represents forward dribbling with switching the hand dribbling the ball. The black curves are the target signals and the cyan curves are the fitted curves computed by optimization. The resulting phases are visualized by the blue bars and the amplitudes are visualized by the opacity of the bars. Top is the results by the evolutionary strategy, where the amplitude and phase information are extracted robustly, and the bottom is gradient-based optimization which yields rather unstable and noisy misalignments.

which together with the optimized amplitude parameter a_i and maximum bone velocity magnitude $||v_\Phi||_\infty$ within a frequency-based frame window $\Phi = \frac{1}{\Delta\phi_i}$ enables to generate distinctive combinations of local 2D phase vectors for each motion (see Fig. $4_{(4,5)}$):

$$\mathcal{P}_i = ||\mathbf{v}_\Phi||_\infty \cdot a_i \cdot \begin{pmatrix} \sin\phi_i \\ \cos\phi_i \end{pmatrix} \in \mathbb{R}^2. \tag{8}$$

The obtained 2D vectors $\mathcal{P}$ for the bones cover information about the timing and speed of the movement, and are fed into the gating neural network as features (see Fig. 2). Their trajectories are smooth as the loss Eq. (4) is defined in a sliding window whose center is the current frame. In particular, modulating the phase vectors by amplitudes has two key aspects in modeling important information about the character motion: First, $\mathcal{P}$ becomes scaled to zero if a bone is not moving, which is important as the phase ϕ_i is rather undefined in such case. Second, it helps to distinctively separate slower and faster movements of similar contact patterns, i.e. walking and running or dribbling at different speeds. Therefore, it can also function as a control variable of the motion; by scaling down the amplitude, the motion of the limb can be inhibited. For example, the motion of the arm opposite to the one dribbling the ball can be reduced by scaling down its corresponding amplitude.

For fitting (minimizing Eq. (4)), we adopt an evolutionary strategy from [Starke et al. 2018]. As our loss function is composed of trigonometric functions, the optimization landscape follows a similar non-convex shape, which results in many local minima. We combine a global probabilistic optimization (genetic algorithm and particle swarm optimization) with a local optimization technique (BFGS). Such a combination can effectively combine the benefits in robustness as well as scalability of evolutionary algorithms with the speed, accuracy and continuity of gradient-based methods. The

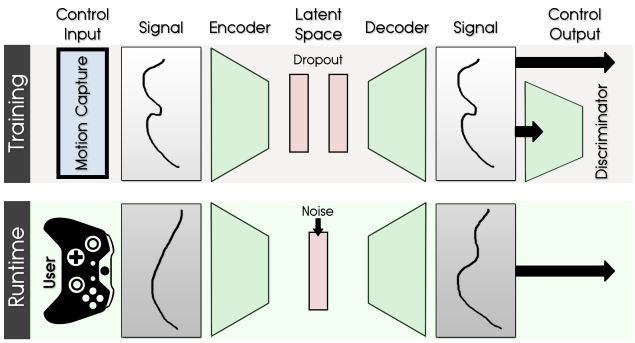

Fig. 6. Overview of the Generative Control Scheme.

optimization pipeline follows that of Starke et al. [2018], except we apply it to finding the phase parameters where they use it to solve an inverse kinematics problem. The global search first stochastically samples a swarm of signal parameters within a range of lower and upper limits. The samples are recombined, randomly offsetted and then resampled by being combined with the best sample in the swarm. The above procedure is repeated for generations to explore and find good samples in the landscape. The BFGS then selectively exploits some elite individuals within subregions to better and more easily converge to solutions of high accuracy. We tested several other gradient-free as well as gradient-based optimization methods, including conjugate gradient, BFGS, Cobyla and Nelder-Mead, which all performed poorly on this task when used alone (see Fig. 5). Our chosen approach can robustly converge to suitable signal parameters.

Our solution can quickly compute a smooth local phase trajectory. Note that we solve the optimization problem per frame and thus the number of parameter is small, which allows the evolutionary strategy to quickly converge to a global minimum. Due to the large overlap of the sliding window, we can obtain a smooth phase trajectory with temporal coherence.

In summary, the local motion phase can be extracted fully automatically from the full body motion and functions as a feature that can produce high quality motion, which we will demonstrate in Section 9.

6 GENERATIVE CONTROL MODEL

We now describe our generative control model, which is introduced to produce a variation of realistic movements from the coarse user control signal. Our idea is to produce a latent space of the control signal via adversarial training of an encoder-decoder network, and then produce a variation by adding noise to the latent code computed from the user-control signal during runtime. The generative control model is pretrained using the control signals computed from the motion capture data.

The generative control model has three components: an encoder **E**, which first takes a smoothed trajectory produced by blending the user gamepad input signal and the autoregressive control signal from the previous time step and encodes it into a latent code; a decoder **G** that produces the control signal from the latent code; finally, a discriminator **D** that distinguishes the generated control signal from the ground truth control signal. The dimensionality of the latent code is set low such that it becomes a bottleneck of the network, and constructs a manifold of the control signal.

More specifically, the encoder **E** is defined as:

$$\mathbf{E}(\mathbf{X}^{\mathcal{V}}) = \mathscr{D}(\mathbf{W}_1^{\mathrm{E}}\, \mathscr{D}(\mathrm{ReLU}(\mathbf{W}_0^{E}\, \mathbf{X}^{\mathcal{V}} + \mathbf{b}_0^{\mathrm{E}})) + \mathbf{b}_1^{\mathrm{E}}), \tag{9}$$

where $\mathbf{W}_0^{\mathrm{E}} \in \mathbb{R}^{h\times n}, \mathbf{W}_1^{\mathrm{E}} \in \mathbb{R}^{m\times h}, \mathbf{b}_0^{\mathrm{E}} \in \mathbb{R}^{h}, \mathbf{b}_1^{\mathrm{E}} \in \mathbb{R}^{m}$ are the network parameters, $h = 512$ is the number of hidden units in each layer, n is the input dimension and m is the dimension of latent space, where we set it to be $n/2$ to create the bottleneck, ReLU is the activation function where we use Rectified Linear Unit, and $\mathscr{D}$ is the dropout layer where the dropout rate is set to be 0.3 to avoid overfitting during training and 0.0 at inference. Similarly, the decoder **G** is defined as:

$$\mathbf{G}(\mathbf{h}) = \mathbf{W}_1^{\mathrm{G}}\mathrm{ReLU}(\mathbf{W}_0^{\mathrm{G}}\, \mathbf{h} + \mathbf{b}_0^{\mathrm{G}}) + \mathbf{b}_1^{\mathrm{G}}. \tag{10}$$

where $\mathbf{W}_0^{\mathrm{G}} \in \mathbb{R}^{h\times m}, \mathbf{W}_1^{\mathrm{G}} \in \mathbb{R}^{n\times h}, \mathbf{b}_0^{\mathrm{G}} \in \mathbb{R}^{h}, \mathbf{b}_1^{\mathrm{G}} \in \mathbb{R}^{n}$ are the network parameters and $h = 512$ is the number of hidden units in each layer. Finally, the discriminator **D** is defined as:

$$\begin{aligned}\mathbf{D}(\mathbf{X}^{\mathcal{V}}) = \mathrm{s}(\mathbf{W}_3^{\mathrm{D}}\mathrm{ReLU}(\mathbf{W}_2^{\mathrm{D}}(\mathbf{W}_1^{\mathrm{D}}\mathrm{ReLU}(\mathbf{W}_0^{\mathrm{D}}\mathbf{X}^{\mathcal{V}} \\ +\mathbf{b}_0^{\mathrm{D}}) + \mathbf{b}_1^{\mathrm{D}}) + \mathbf{b}_2^{\mathrm{D}}) + \mathbf{b}_3^{\mathrm{D}}).\end{aligned} \tag{11}$$

where $\mathbf{W}_0^{\mathrm{D}} \in \mathbb{R}^{h_0\times n}, \mathbf{W}_1^{\mathrm{D}} \in \mathbb{R}^{h_1\times h_0}, \mathbf{W}_2^{\mathrm{D}} \in \mathbb{R}^{h_2\times h_1}, \mathbf{W}_3^{\mathrm{D}} \in \mathbb{R}^{1\times h_2}$, $\mathbf{b}_0^{\mathrm{D}} \in \mathbb{R}^{h_0}, \mathbf{b}_1^{\mathrm{D}} \in \mathbb{R}^{h_1} \mathbf{b}_2^{\mathrm{D}} \in \mathbb{R}^{h_2} \mathbf{b}_3^{\mathrm{D}} \in \mathbb{R}^{1}$ are the network parameters, h_0, h_1 and h_2 are the number of hidden units in each layer set to 128, 64, 32, respectively, and s is the sigmoid function to map the output to be a probability between 0.0 and 1.0.

The system is trained with reconstruction loss and adversarial loss. We use $\mathbf{X}^{\mathcal{V}}$, the control variables of the input vector **X**, as our training data. It is encoded in to the latent space by the encoder: $\mathbf{h} = \mathbf{E}(\mathbf{X}^{\mathcal{V}})$. After the encoder, the original control signal is reconstructed by the decoder $\mathbf{X}_r^{V} = \mathbf{G}(\mathbf{h})$. The reconstructed signal is evaluated by a $L1$ loss with the original signal $L_{\ell_1}(\mathbf{X}^{\mathcal{V}}, \mathbf{X}_r^{V})$ and the adversarial loss $L_{adv}(\mathbf{E}, \mathbf{G}, \mathbf{D})$:

$$L_{all} = L_{adv} + \lambda_{\ell_1} L_{\ell_1}, \tag{12}$$

where λ_{ℓ_1} is a weight set to 5.0 to make sure that the L_{adv} and L_{ℓ_1} have similar magnitude. The adversarial loss is defined by

$$L_{adv}(\mathbf{E}, \mathbf{G}, \mathbf{D}) = E_{real}\left[\log(s_{real})\right] + E_{fake}\left[\log(1 - s_{fake})\right], \tag{13}$$

where we denote the output score of the discriminator **D** on real and fake inputs by s_{real} and s_{fake}, respectively. The encoder, decoder and discriminator are trained jointly.

During runtime, $\mathbf{X}^{\mathcal{V}}$ is computed by blending $\mathbf{Y}^{\mathcal{V}}$ in the previous cycle with the user input signals (see Eq. (1)). Then it is projected to the latent space, and modified by adding a random noise sampled from a Gaussian. With this, the control signal recovered by the decoder will be containing valid variance which is very hard to be hand-coded from gamepad input. Note that there is a tradeoff between the scale of the noise and the response time of the character: the larger the scale is, the less responsive the character becomes due to the deviation from the user control signal.

We show some examples of an input and output control signal in Fig. 7. It can be observed that the input trajectory is a smooth trajectory while the output trajectories produced from different