Exploiting Novel Deep Learning Architectures in Character Animation Pipelines

Saeed Ghorbani

A dissertation submitted to the Faculty of Graduate Studies
in partial fulfilment of the requirements
for the degree of

Doctor of Philosophy

Graduate Program in
Electrical Engineering and Computer Science
York University
Toronto, Ontario

April, 2022

Abstract

This doctoral dissertation aims to show a body of work proposed for improving different blocks in the character animation pipelines resulting in less manual work and more realistic character animation. To that purpose, we describe a variety of cutting-edge deep learning approaches that have been applied to the field of human motion modelling and character animation.

The recent advances in motion capture systems and processing hardware have shifted from physics-based approaches to data-driven approaches that are heavily used in the current game production frameworks. However, despite these significant successes, there are still shortcomings to address. For example, the existing production pipelines contain processing steps such as marker labelling in the motion capture pipeline or annotating motion primitives, which should be done manually. In addition, most of the current approaches for character animation used in game production are limited by the amount of stored animation data resulting in many duplicates and repeated patterns.

We present our work in four main chapters. We first present a large dataset of human motion called MoVi. Secondly, we show how machine learning approaches can be used to automate proprocessing data blocks of optical motion capture pipelines. Thirdly, we show how generative models can be used to generate batches of synthetic motion sequences given only weak control signals. Finally, we show how novel generative models can be applied to real-time character control in the game production.

Acknowledgements

This document owes its existence to many people who have offered advice and support throughout its development. First and foremost, I would like to thank my supervisor, *Niko Troje*, for his continuous guidance. Niko, your broad knowledge and commitment to scientific excellence have been instrumental in my development as a scientist, and I am genuinely proud to have trained under you.

My family has also given unending support. To my wife, Kimia, you are a vital steadying presence in my life, and you have provided me with an invaluable logical sounding board for my ideas and arguments. Your love and advice have been instrumental in allowing me to achieve my goals. To my parents and sister, your cheers and backdrop of love and support have been appreciated more than I know how to say. Thank you to my collaborators, who have continuously helped me find new ideas. In particular, I would like to thank Marcus Brubaker and Ali Etemad. Marcus, you provided valuable and insightful discussions on the topic of generative models, and I am grateful to you for pushing my ideas on the topic to new heights. Ali, you gave me the opportunity to start this whole journey in the first step. I will always be grateful for this and the precious advice you provided on the topic of human motion modelling. While the list of additional colleagues is too numerous to mention everyone, I hope you all know how much I have valued your input and critiques.

To my committee, Marcus Brubaker, Michael Brown, Joel Zylberberg, and my external examiner, Michael Greenspan, I would like to sincerely thank you for your time, ideas, and helpful feedback. You helped push me and have made this dissertation better and more complete. I would like to acknowledge the opportunity given by Ubisoft La Forge, to work in such a great and knowledgeable team as an Intern. Finally, I would also like to acknowledge the financial support I

received from the National Sciences and Engineering Research Council (NSERC).

Contents

Abstract ii

Acknowledgements iii

Contents v

List of Tables ix

List of Figures xi

List of Abbreviations xvii

1 Introduction 1

1.1 Organization of Dissertation 4

2 Background 6

2.1 Human Motion Recording, Representation, and Processing 7

2.1.1 Motion Capture Systems 7

2.1.2 Pose Representation 11

2.1.3 Preparing Motion Sequences for Modelling Pipelines 14

2.2 Human Motion Modelling 19

2.2.1 Deterministic Models 23

2.2.2 Probabilistic Models 35

2.2.3 Evaluation Methods 60

3 MoVi: A Large Multi-Purpose Human Motion and Video Dataset 63

3.1 Preface . . . 64

3.2 Introduction . . . 64

3.3 Methods . . . 66

3.3.1 Subjects . . . 66

3.3.2 Acquisition Setup . . . 66

3.3.3 Data Collection . . . 69

3.3.4 Preprocessing . . . 72

3.3.5 Calibration . . . 72

3.3.6 Synchronization . . . 75

3.3.7 Skeleton and Body Shape Extraction . . . 76

3.4 Data Records . . . 77

3.4.1 Raw data . . . 79

3.4.2 Processed data . . . 79

3.5 Applications . . . 79

3.6 Code availability . . . 80

3.7 Conclusion . . . 81

4 Auto-labelling of Markers in Optical Motion Capture by Permutation Learning 82

4.1 Preface . . . 83

4.2 Background . . . 83

4.3 Proposed Framework . . . 86

4.3.1 Data Preprocessing . . . 87

4.3.2 Permutation Learning Model . . . 88

4.3.3 Trajectory Labelling . . . 91

4.4 Experiments and Evaluations . . . 93

4.4.1 Data . . . 93

4.4.2 Training . . . 94

4.4.3 Permutation Learning Model Evaluation . . . 95

4.4.4 Trajectory Labelling Evaluation . . . 96

4.5 Conclusion . . . 99
4.6 Future Work . . . 100

5 Probabilistic Character Motion Synthesis using a Hierarchical Deep Latent Variable Model 101

5.1 Preface . . . 102
5.2 Introduction . . . 103
5.3 System Overview . . . 105
5.3.1 Data Preprocessing . . . 107
5.3.2 Hierarchical Probabilistic Recurrent Network . . . 108
5.3.3 Motion Cell . . . 109
5.3.4 Attributes Classifiers . . . 113
5.3.5 Objective Function . . . 114
5.4 Implementation and Training . . . 119
5.4.1 Dataset . . . 119
5.4.2 Training Process . . . 119
5.5 Experiments and Evaluation . . . 121
5.5.1 Sequence Samples . . . 121
5.5.2 Models and Ablations . . . 123
5.5.3 Quantitative Evaluation . . . 123
5.5.4 Qualitative Evaluation . . . 125
5.6 Conclusion . . . 127
5.7 Future Work . . . 127

6 KinFlow: A Probabilistic Approach for Multi-Modal Character Control 129

6.1 Preface . . . 130
6.2 Introduction . . . 130
6.3 System Overview . . . 132
6.4 Preparing Dataset . . . 134
6.4.1 Motion Capture Datasets and Extracting Control Parameters . . . 134
6.4.2 Character State, Conditioning Input, and Control Parameters . . . 136

6.5 KinFlow Architecture . . . 138
6.5.1 Normalization Layer . . . 139
6.5.2 1×1 Convolution Layer . . . 140
6.5.3 Coupling Layer . . . 142
6.5.4 Temporal Affine Layer . . . 143
6.5.5 Multi-scale Architecture . . . 145
6.5.6 Objective function . . . 145
6.6 Implementation and Training . . . 146
6.6.1 Data processing . . . 146
6.6.2 Model Parameters . . . 147
6.6.3 Training Process . . . 147
6.6.4 Runtime . . . 147
6.7 Results . . . 148
6.7.1 Experiments . . . 148
6.7.2 Quantitative Evaluation . . . 151
6.8 Conclusion . . . 154
6.9 Future Work . . . 154

7 Conclusions and Future Directions **156**
7.1 Summary of Contributions . . . 157
7.2 Future Directions . . . 159
7.2.1 Towards Motion Capture Data Processing . . . 159
7.2.2 Towards Motion Modelling and Character Animation . . . 160

Bibliography **161**

List of Tables

3.1 Participant characteristics of the 60 women and 30 men. 67

3.2 Overview of the five different capture rounds. F = full motion capture markerset, S = sparse motion capture markerset + IMU, I = IMU; *1 = with rest A-pose, *2 = without rest A-pose. 71

3.3 Naming conventions and structure of all files available from the database. ⟨`ID`⟩ ∈{`1,2,...,90`} indicates the subject number, ⟨`round`⟩ ∈{`F,S1,S2,I1,I2`} the data collection round, and ⟨`camera`⟩ ∈{`PG1,PG2,CP1,CP2`} the camera type where PG indicates the computer vision cameras and CP the cellphone cameras. 78

4.1 A comparison of performance of different models in labelling a single test frame as an initialization step in the presence of a varying number of occlusions in the test frames (show in each column). The first and second rows illustrates the performance when the model is trained with and without occlusions, respectively. Third row, shows the results when the Sinkhorn layer is replaced by a Softmax function. It can be seen that the model trained on occlusion augmented training set outperforms the rest when having occlusions in the frames. 96

4.2 The performance of trajectory labelling with different settings of p and q in our scoring function. 98

5.1 The architecture of model components. FC(n) is the abbreviation for Fully Connected linear layer with n nodes. Conv1D and ConvTranspose1D are one-dimensional convolution and transposed convolution layers. AdaptiveAvgPool1D is one-dimensional adaptive average pooling layer. 120

5.2 Results from quantitative evaluations using IS (higher score is better) and FID (lower score is better). 125

5.3 Results for qualitative evaluation. The results show the average scores assigned to a set of motions sampled from each action set and from each model, where 1 corresponds to completely unrealistic and 10 corresponds to completely realistic. . . 126

6.1 Comparing different models for modelling motion data in terms of negative-log-likelihood (NLL) per dimension on the held-out test data, training duration, the number of learnable parameters, and runtime speech. 153

List of Figures

1.1 The basic pipeline for data-driven character animation frameworks 3

2.1 Motion Capture Systems. 8

2.2 An overview of optical motion capture pipeline 9

2.3 The hierarchical movement generation process. 19

2.4 The Encoder-Recurrent-Decoder Architecture proposed by Fragkiadki et al. [1]. The pose space is transformed into an intermediate representation space using *Encoder* where the dynamics are learned by *Recurrent* network. Then, the output is reconstructed by transcribing the output of Recurrent network to the pose space by *Decoder* network 28

2.5 Seq-2-Seq Architecture proposed by Martinez et al. [2]. The ground truth initial sequence is fed to the *encoder*, and joint angle velocities of predicted sequence is computed by the *decoder* and linear networks that takes its own sample as input during training. 28

2.6 The architecture proposed by Holden et al. [3]. Editing and synthesizig motions is performed in a learned motion manifold provided by a pre-trained bi-directional autoencoder. The control network maps high-level control parameters to the corresponding motion represented in the learned manifold 31

2.7 The architecture of Phase-Functioned Neural Network proposed by Holden et al. [4]. The phase function takes phase of the locomotion as input and modulates the weights of the autoregressive network. Providing the phase explicitely to the model helps it to disambiguate the next pose. 33

2.8 The Mode-Adaptive Neural Network architecture proposed by Zhang et al. [5] composed by Prediction Network and Gating Network. Gating Network receives the current state as input and modulates the weights of the prediction network by blending the learnable expert weights. Each expert is specialized for a subdomain in the training data space. 34
2.9 Graphical representation of Boltzmann Machines and Restricted Boltzmann Machines. 38
2.10 (a) cRBM architecture [6] where both current visible state and hidden state are conditioned on previous visible state to model the temporal dependencies. (b) FCRBM architecture [7] as an extension of cRBM where the interactions are gated by style features. 39
2.11 The graphical representation of generative adversarial networks architecture proposed by Goodfellow et al. [8]. 42
2.12 The graphical representation of architecture proposed by Gui et al. [9]. The final objective function comprises two adversarial losses (corresponding to two discriminators) and the geodesic loss. 43
2.13 Visualization of a stack of dilated causal convolutional layers in Wavenet proposed by Van Den Oord [10]. 46
2.14 An overview of the original VAE model proposed by Kingma et al. [11]. 49
2.15 An overview of MotionVAE proposed by Ling et al. [12] 51
2.16 An overview of conditional RNN-VAE model proposed for synthesizing head motion given speech data by Greenwood et al. [13]. During the generation phase, sampling from prior adds natural variations to the generated head movements while being controlled by the conditioning audio. 52
2.17 An overview of conditional VRNN model at one time-step proposed by Chung et al [14]. f_{enc}, f_{dec}, f_z, f_{prior} are arbitrary functions such as neural networks. 53
2.18 Block diagram of Affine Coupling layer 57
2.19 Stacking several Flow Steps resulting in higher expressivity 57

2.20 The coupling flow structure used in MoGlow, WaveGlow, and FlowWaveNet. The conditioner was defined by and LSTM for MoGlow and by a WaveNet architecture for WaveGlow and FlowWaveNet. 58

3.1 Top view sketch of the capture room set-up. The position of the video cameras and motion capture cameras were arranged to cover a space of approximately 3 by 5 meters. 68
3.2 Example pictures of one female and one male actor wearing the IMU suits used for the capture rounds S1, S2, I1, and I2. 69
3.3 Placement of IMU sensors on the body. 70
3.4 Front and side view of aligned video frame, joint locations, and estimated body mesh (computed by MoSh++) for one female and male participant. 74

4.1 A sample of list of unlabelled trajectories after a capture sessions 84
4.2 An overview of our proposed framework: the input to our system is a sequence of unlabelled (shuffled) 3D trajectories. After the preprocessing stage, the label for each marker is estimated by applying permutation learning to each individual frame. The resulting labelled frames are then concatenated to form trajectories again. These trajectories are then used as input to the trajectory labelling stage where a temporal consistency constraint is used to correct mislabelled markers 86
4.3 We make the data invariant to the orientation of the subject by aligning the first and second PCA components of the cloud of the markers to the y and z axes, respectively. 87
4.4 Formulating the labelling task as a permutation learning model. Unlabelled markers can be considered as shuffled version of sorted labels by a permutation matrix (figure a). Labelling task is then can be defined as finding this permutation matrix given the 3D location of the markers and applying its transpose on the shuffled data to sort the makers based on a predefined order of labels (figure b). 88

4.5 An overview of our permutation learning model: In (a), the learning phase is depicted, where the parameters of the learning module are optimized to minimize the multi-class, multi-label cross-entropy loss. In (b), the running phase is illustrated, where the Hungarian algorithm is applied to the outputted DSM to estimate the optimal permutation matrix which is then applied to the shuffled labels to sort them to the predefined order. 90
4.6 Schematic diagram of the feed-forward deep residual neural network used for permutation learning. 94
4.7 Plot of accuracy-precision curves for labelling frames without and with occlusions. In the latter case an average of 2.5 markers was missing in each frame. 97
4.8 A comparison between simulated occlusions for our trajectory labelling evaluation and real scenario for the 41 marker layout. In both figures, the occlusion ratio is 4%. In, (a) the markers and frames are occluded with a uniform probability. In (b), we searched for an equal-length window with the same occlusion ratio in the real data. The average trajectory length in simulated occlusions is much higher than the real scenarios. 99

5.1 Samples of a real motion sequence (blue) and synthesized motion sequence generated by our model (orange) 102
5.2 An overview of our recurrent model. During training, information is projected into high level feature vectors via f^{w}_{enc}. Motion Cells operate on these feature vectors in latent space to combine information from the preceding sequence with stochastic variability in order to output the next step in the sequence. This output is converted to a joint angle representation via f^{w}_{dec}. Normalization ensures that the representations fall within valid ranges. 106
5.3 Internal structure of Motion Cell. A Motion Cell can be viewed as a recurrent unit conditioned on control signals. 109
5.4 Scheduling the coefficients for each term in Eq. 5.14 during training. 121
5.5 Samples of real (blue) and synthetic (orange) motion sequences. 122

5.6 Visualizing the activations of the last layer of action classifier projected onto two dimensions using t-sne for real data (circles) and data synthesized data (crosses) by our model. As can be seen, synthesized samples strongly coincide with the corresponding clusters formed by real data. 127

6.1 An overview of KinFlow. At each step, a sample from base distribution is mapped to the current character state while being conditioned on the control signal and previous states by modulating all invertible transformations. 132

6.2 A visualization of the character state parameterization of our system. The pose of the character at each time-step is defined by joint transformations in addition to joint velocities (light blue arrows). Sub-sampled predicted trajectory positions and directions are shown in white markers and black arrows, respectively. Gait phase is also predicted at each frame 137

6.3 The proposed 1×1 convolution layer. At each step the weight matrix for 1×1 convolution is computed as a function of modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$ and the previous input the this layer x_t. 141

6.4 The coupling flow layer is used in our model. We modelled the conditioner using a GRU Cell and conditioned that to the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$ in addition to the first data partition $x_t^{1:d}$ 143

6.5 The temporal affine layer is used in our model. The input is modulated by the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$ and the input in the previous frames $x_{<t}$ using a GRU. . . 145

6.6 Top: Height of the left foot (in centimetres) for two generated samples from the exact same input control. Bottom: The difference between the left foot heights of the two samples. 149

6.7 Top: The real part of the left arm rotation for two generated samples from the exact same input control. Bottom: The difference between the left arm rotations of the two samples. 149

6.8 Projected root onto the horizontal plane for two generated samples from the exact same input control. 150

6.9 Results showing our character crouching 150

6.10 Results showing our character jumping. 151
6.11 Results showing our character controlled by adjusting the future trajectory position and direction and action type. 151
6.12 Controlling our character by predefined paths. 152
6.13 Samples of character moving in the walkable areas. The environment is baked for categorizing objects using Unity Navmesh for obstacle avoidance. 152

List of Abbreviations

AdaIN	Adaptive Instance Normalization
AGED	Adversarial Geometry-aware Encoder-Decoder
AMASS	Archive of Motion capture As Surface Shapes
BFE	Basis Function Expansion
BMI	Body Mass Index
CNN	Convolutional Neural Network
cRBM	conditional Restricted Boltzmann Machines
C3D	Coordinate 3D
DBN	Deep Belief Network
DLVM	Deep Latent Variable Model
DoF	Degree of Freedom
DSM	Doubly-Stochastic Matrix
ELBO	Evidence Lower BOund
ERD	Encoder-Recurrent-Decoder
FCRBM	Factored Conditional Restricted Boltzmann Machines
FID	Fréchet Inception Distance Score
FPCA	Functional Principal Component Analysis
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Models
GP	Gaussian Process
GPLVM	Gaussian Process Latent Variable Models
GPDM	Gaussian Process Dynamics Model

GRU	Gated Recurrent Units
HMM	Hidden Markov Model
IMU	Inertial Measurement Unit
IS	Inception Score
KL	Kullback Leibler
LSTM	Long-Sort Term Memory
MANN	Mode-Adaptive Neural Networks
MAP	maximum a posterior
MDP	Markov Decision Process
MLP	Multi-Layer Perceptrons
MoE	Mixture of Exports
MoSh	MOtion and SHape capture from sparse markers
NPC	Non-Player Character
NSM	Neural State Machine
PCA	Principal Component Analysis
PFNN	Phase-Functioned Neural Network
P3P	Perspective-Three-Point
QTM	Qualisys Track Manager
RBM	Restricted Boltzmann Machines
RNN	Recurrent Neural Network
RVAE	Recurrent Variational AutoEncoder
SGPLVM	Scaled Gaussian Process Latent Variable Model
SHMM	Stylistic Hidden Markov Models
SLERP	Spherical Linear Interpolation
SMPL	Skinned Multi-Person Linear model
SPL	Structured Prediction Layer
SQP	Sequential Quadratic Programming
S-RNN	Structural- Recurrent Neural Networks
T-SNE	T-distributed Stochastic Neighbor Embedding

VAE	Variational AutoEncoder
VLB	Variational Lower Bound
VRNN	Variational Recurrent Neural Network
WGAN	Wasserstein GAN

Chapter 1

Introduction

The rapidly growing dynamic and interactive media such as video games has resulted in an increase in the need for high-quality and diverse content for a better engaging experience. However, poor and unrealistic animation can disengage the audience and not convey the intentions, feelings, and emotions. On the other hand, good character animation can make the audience truly absorb the details, sympathize, and relate to the character and story.

Over the past two decades, data-driven tools for the synthesis and production of character animation have made significant success in the graphics community. In particular, some of these techniques have been widely adopted in the video game industry as they allow an automatic production of character animation where players, animators, or designers specify only a high-level description of the required animation.

The base pipeline for the data-driven frameworks is shown in Figure. 1.1. Each block in this pipeline is a major phase in the framework and contains several sub-steps. The first block includes prototyping and designing the model based on the general goal, system requirements, and available resources. The next step is to collect the required motion data. Planning and prototyping the data collection phase entails a variety of decisions and choices, such as the number of actors and type of motions made based on the model design and other influential factors. Then the collected data are cleaned and prepared to be used in the animation framework. The required processing steps in this block heavily depend on the model design. Finally, the designed model is implemented and utilized along with the prepared data to produce character animation. Despite significant successes in these frameworks, there are still many gaps and challenges in each phase of these frameworks which need to be addressed. In the following, we will explain some of the gaps that motivated our main contributions in this thesis.

First, since the data collection step is a costly process, the collected datasets are usually limited to a small number of actors and/or motions or a subspace of idiosyncratic styles. This limitation results in a low diversity in the produced motion primitives in addition to low controllability and flexibility of the animation frameworks. This issue also hampers the progress of learning models due to the sparsity of provided training data when using machine learning. Some efforts have been made to alleviate this issue by unifying datasets via re-processing raw motion capture data [15] or re-targeting motion data into a unified skeleton [3]. However, such strategies pose their own limitations and challenges as well.

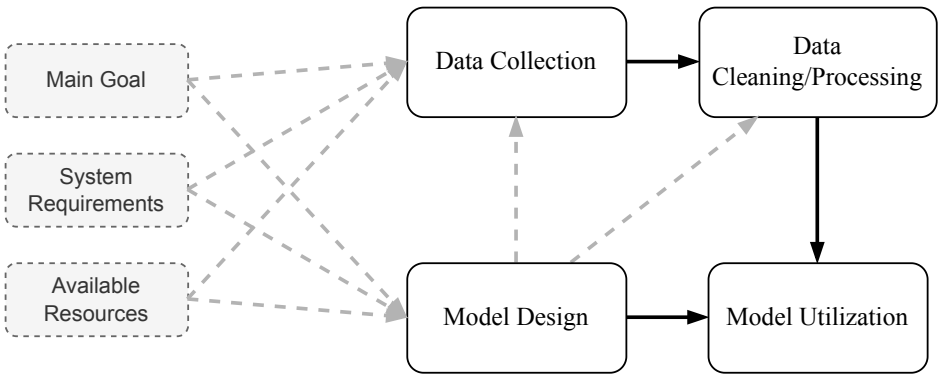


Figure 1.1: The basic pipeline for data-driven character animation frameworks

Second, while the motion capture step is a long and labour-intensive process by itself, in the current production pipelines, the recording phase is followed by tedious cleaning and processing steps such as marker labelling in the motion capture pipeline, cleaning data, and offline key-framing of characters which must be done manually. Such steps in game production should be done with extreme scrutiny and accuracy to achieve the desired outcome. Therefore, a large amount of the time of motion capture technicians and skilled animators is spent on these blocks, turning them into the main friction points of the game production pipeline. Automating these blocks and minimizing the amount of needed manual work can significantly speed up the overall pipeline process in addition to reducing the amount of human error.

Third, the current designed models for character animation such as Motion Graphs [16], and Motion Matching [17] which are heavily used in the game production, are limited by the memory usage, which scales linearly with the amount of stored animation data. In addition, these approaches must search for a suitable match at runtime, which can be slow for a large motion capture dataset. Consequently, with a limited size of datasets, there will be a lot of duplicates and repeated patterns in the resulting character movements, which our visual system can detect after a short time. Therefore, there has been a trade-off between the diversity of the character movements and the computational and memory budget. Due to these limitations, neural network generating models have attracted more attention. These models are scalable with the data and are capable of learning expressive motion representation and control parameters. However, such models are not yet prevalent in the production pipelines as they are less flexible and usually need higher processing time in the runtime. In addition to mentioned challenges, one of the main challenges in these

models is integrating natural variations such as natural stochasticity and style presented in human movements, which is a crucial component to having a natural and convincing synthesized motion.

1.1 Organization of Dissertation

My dissertation will be formed by binding together several studies contributing to data-driven character animation pipelines by addressing some of these challenges. The contribution of the proposed thesis can be presented in four chapters.

In Chapter 3, we address the lack of a suitable large human motion dataset by collecting a large multipurpose human motion and video dataset (MoVi), which contains motion data recorded from a large number of actors, action types, and different styles. In addition, to help research in pose estimation and body shape reconstruction from image and video data, the motion capture data were collected along with the frame-wise synchronized and cross-calibrated video data recorded from different points of view.

In Chapter 4, we propose a data-driven approach to automate the marker labelling of optical motion capture data. Marker labelling is one of the main time-consuming, labour-intensive, and error-prone tasks of optical motion capture data processing and animation pipelines. The proposed method uses a permutation learning model for initializing the motion capture data labels and then utilizes temporal consistency to identify and correct remaining labelling errors. Our framework can be reliably run in real-time that potentially resulting in faster, cheaper, and more consistent data processing in motion capture pipelines.

In Chapter 5, we present a probabilistic framework to synthesize character animations based on weak control signals, such that the produced motions are realistic while retaining the stochastic nature of the human movement. Weak control signals are particularly useful for tasks in which a large number of motion primitives are required without a need to a high level of supervision or user control. In these situations, requiring strong control signals such as motion trajectory would be unnecessary or even impossible at runtime. Our framework can continuously generate batches of novel motion clips in runtime and by pre-determined control attributes. Therefore, this Chapter's contribution is two-fold by addressing the first and third challenges mentioned above. First, it can be used for generating synthetic motion data for other frameworks such as our proposed framework in

Chapter 6 or Motion Matching. Second, it is capable of producing high-quality, realistic, and diverse motion primitives with a low level of supervision and a high level of diversity for the tasks such as crowd simulation. The proposed architecture, which was designed as a hierarchical recurrent model, maps each sub-sequence of motions into a stochastic latent code using a variational autoencoder extended over the temporal domain. We also proposed an objective function that respects the impact of each joint on the pose and compares the joint angles based on angular distance. We used two novel quantitative protocols and human qualitative assessment to demonstrate our model's ability to generate convincing and diverse periodic and non-periodic motion sequences a the need for strong control signals.

In the last Chapter (Chapter 6), we propose a probabilistic approach for character control which generates realistic character animation with natural stochasticity and variations while being controlled by a gamepad and based on the conventional game production frameworks. We propose a flow-based architecture trained on the collected data in Chapter 3 and synthetic data generated by our model proposed in Chapter 5. Compared to previous flow-based models for character animation, our framework has achieved better quantitative and qualitative results, has a runtime computing speed of 60 frames per second which is compatible with the current game engine requirements, can be controlled by a conventional gamepad control system, is probabilistic not only in modelling of the pose of the character but also the global movement and predicted trajectories. These improvements were achieved by careful design strategies in model architecture and motion characterization.

Chapter 2

Background

This chapter provides some background on the main blocks of data-driven character animation pipelines (see Figure 1.1). In the first section, we will review the data collection and data preprocessing steps in the motion capture setups with a focus on optical motion capture systems with passive markers. In addition, a review of the common steps for preparing the processed data to be used in data-driven approaches will be provided. Section 2 reviews the data-driven models for character animation with a focus on statistical methods.

2.1 Human Motion Recording, Representation, and Processing

2.1.1 Motion Capture Systems

Traditionally, character animation was a long and labour-intensive process involving skilled artists posing a virtual character at a high frame rate using tools and software such as Maya and 3D Studio Max. However, the ever-increasing demands on film and game production for more realistic animations, less manual work, and more robust and faster pipelines have fueled more interest in finding better and more efficient approaches for character animation. While the use of key-framing, inverse kinematics, or physics-based models has improved the process of traditional character animation, the overall process is prolonged and expensive, as it requires many highly skilled and professional animators. Therefore, to meet the demands of more automatic character animation pipelines, the film and game industries resorted to motion capture technologies in an attempt to pave the way for modern character animation. By using motion capture systems, a significant amount of manual and time-consuming frame-by-frame character posing tasks was removed by directly recording the actors performing the required actions. In addition, using directly recorded samples significantly increased the quality of produced animations. Motion capture systems track the motion of a number of sensors in a 3D space over time, which are ultimately translated to the 3D dimensional representation of a subset of main joints in the character skeleton and/or body surface. These systems offer many advantages, such as low latency and providing information for extracting body shape. There are also a few disadvantages to point such as high cost, space limit, tedious post-processing, and not capturing constraints imposed by physics. Classification of motion capture systems is demonstrated in Figure 2.1

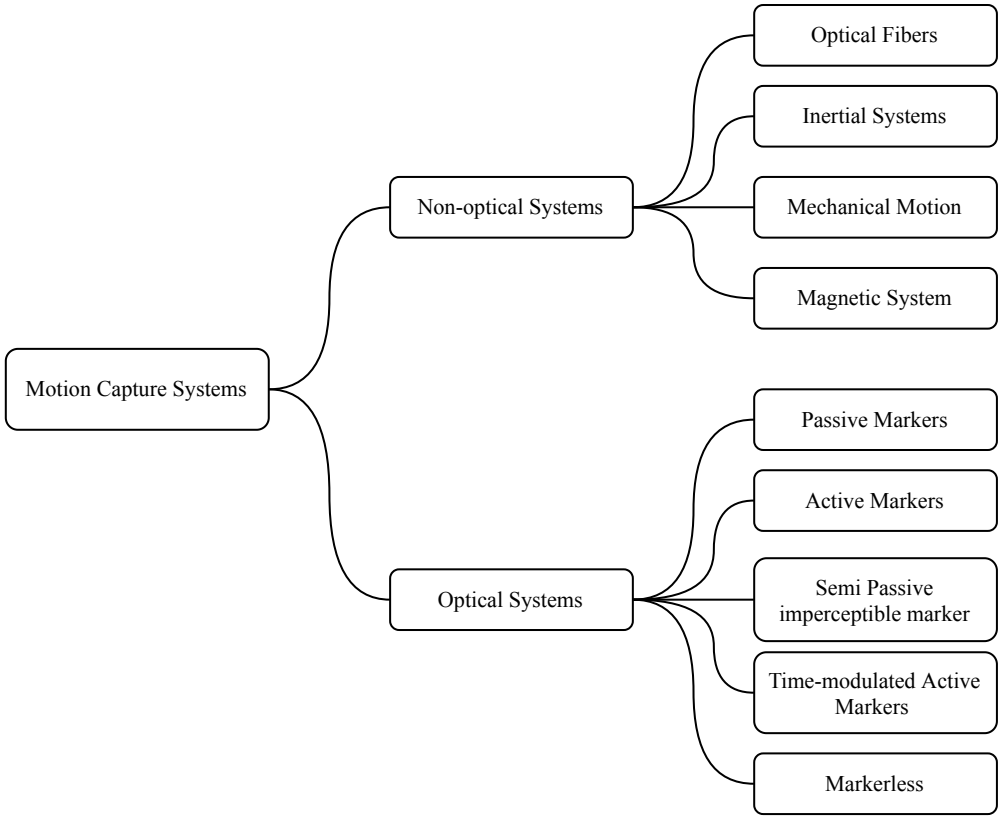


Figure 2.1: Motion Capture Systems.

2.1.1.1 Optical Motion Capture Systems

The optical motion capture system is one of the most popular motion capture systems, which is widely used in animated movies, video games, biomechanics, and health care industries. Each capture session in optical motion capture consists of tracking and recording the motion of a set of optical retro-reflective markers that are strategically positioned on the body parts. Given the labelled trajectories of the markers, their Cartesian positions are used to compute a set of kinematic and/or kinetic measurements using a predefined calculation model [18, 19, 15]. The kinematics include joint positions, joint angles, joint positional or rotational velocity and acceleration, etc. The kinetics include joint force, joint moment, joint power, rotational or translational scaler energy of a segment, potential energy, linear or angular momentum, etc.

Optical motion capture was originally used to record the skeleton's motion as a proxy for human movement. Thus, the markers were attached to the parts with minimal noise caused by soft-tissue micro motions. However, in some later approaches [19, 15], soft-tissue and subtle surface motions

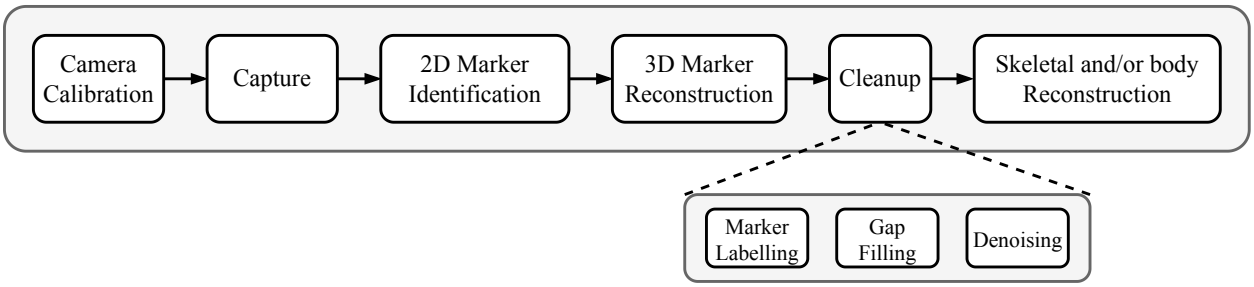


Figure 2.2: An overview of optical motion capture pipeline

were also recorded to estimate the body shape and dynamic shape deformations.

A typical motion capture pipeline is depicted in Figure 2.2. In the following, we briefly describe each block.

Calibration: A calibration process links the cameras to the real environment (including each other). The primary purpose of this stage is to create a relative 3D reference for each camera so that each camera has its own reference in 3D space relative to the other cameras and a single world coordinate space. The origin of the world coordinate system and its orientation in the room is usually defined by placing an L-frame with attached markers located on the ground in the center of the capturing space. The relative distances of the cameras are referenced with the world origin as it is in the actual room. The calibration is performed by recording the motion of a wand with attached markers covering the capturing space and for more than a minimum number of required frames. The location and orientation of the cameras relative to the world coordinate system and each other are found using a Perspective-3-Point (P3P) algorithm.

Capture Session: During the recording step, the motion of passively reflecting markers, which are attached to the body according to a predefined marker layout, are tracked by the calibrated cameras. The actions are performed based on a predefined scenario or given instructions to the actor. For example, for an accurate body pose/surface estimate, actors usually wear tight outfits, or the markers are attached to the skin.

2D marker identification and 3D Marker Reconstruction: During capture, the position of the markers is computed in two steps. First, the 2D location of each marker is recovered in the field of view of each camera. The optical motion capture cameras are equipped with infrared LEDs and an infrared band-pass filter over the lens to filter out non-reflective materials. Then, given the

2D location of markers in the field of view of at least two cameras, the 3D position of markers is computed from 2D data recorded by each of the calibrated cameras using triangulation techniques. This process results in a set of unlabelled 3D trajectories listed in random order. Each trajectory represents the motion of a single marker in terms of its 3D position over time.

Cleanup: Cleanup is the manual and automatic postprocessing steps required before skeletal or body reconstruction. Typically, this stage includes Marker Labelling, Gap Filling, and Denoising, briefly described below.

Marker Labelling: One of the main steps after the recording phase is to label each trajectory, defined as assigning it to a specific body location defined by a label or ID. Most parts of this process should be performed manually and can be very time-consuming and, therefore, a friction point for many motion capture systems. The current motion capture software offers integrated models to make this step semi-automatic. However, since the used approaches are based on anthropometric measures and rigid segment detection, an initialization step should be done manually for each actor. This is still an active area, and many approaches have been proposed to automate this stage [20, 21, 22, 23].

Gap Fillings: Usually, the motion of all markers cannot be recorded completely over the capture time. This is due to various reasons, such as occlusions, noise, and ghost markers. The integrated optical motion capture software provides in-built interpolation functionality for the short gaps in the marker trajectories. But, filling in between the large gaps is a non-trivial task, and the conventional interpolation techniques may induce some inaccuracy into the model. Biomechanical applications that rely on robust and accurate measures consider large gaps as non-negligible artifacts and minimize them by a careful design of motion scenario and camera mounting layout. However, such artifacts are usually neglected to some extent in computer vision and computer animation applications. For example, in MoSh [19] and MoSh++ [15] the body mesh is first estimated given a set of visible markers. Then the joint positions are regressed given the estimated body mesh. In this way, the position of all joints is estimated even in the presence of occlusions, while the accuracy of such estimation depends on the number of occlusions at each frame.

Denoising: The raw optical motion capture data may include unwanted noise such as occlusions, mislabeled and ghost markers, jitter, and high-frequency component imposed by different internal

and external sources. Usually, the motion capture software provides some in-built denoising functionalities such as low-pass frequency filters to filter out the high-frequency noise component. In addition, extra denoising approaches can be integrated into the pipeline for a more robust reconstruction. For example, Holden et al. [24] proposed a data-driven approach for directly inferring joint transforms from raw mocap data using a neural network architecture. The proposed network is trained on a per-pose basis where given an input pose from training data, a set of virtual markers are reconstructed using a linear blend skinning and based on a predefined marker configuration. Then a noise function is applied to corrupt the virtual marker data by randomly removing or moving the virtual markers. Finally, a denoising feed-forward neural network is trained to map the corrupted marker data to the original joint transforms. In another work, we used a one-dimensional convolutional denoising autoencoder which was directly applied to the motion sequences [25].

Skeletal and body reconstruction: Given labelled marker trajectories, one can reconstruct the pose at each frame by computing the joint locations and orientations based on a predefined skeletal model. This can be done by formulating each element as a biomechanical function of marker locations, and anthropometric data [18] or using a data-driven approach [19, 15]. In addition to pose, some approaches estimate body mesh using statistical body models. Loper et al. [19] formulated the body shape parameters estimation from raw mocap data as a *generative inference model*, where model parameters are optimized such that the model output is similar to real-world observations. To this end, the parameters of a statistical body model were optimized by minimizing the distance between the reconstructed virtual markers on the estimated body and the ground-truth location of the real markers.

2.1.2 Pose Representation

The previous section described how the optical motion capture pipeline processes the raw marker trajectory data and provides pose data over time. In this section, we will have a review of the common ways of representing pose in the lowest level of abstraction suitable to be stored or ready to be animated.

Human motion can be represented as a sequence of poses where each pose represents the skeletal configuration at a specific time step. The skeleton is an articulated figure as a structure that consists

of a series of rigid links connected at joints. Since we are dealing with motion as a sequence of poses and hence their kinematics, we focus on how to represent pose at each time frame. Having defined the pose for each time-frame, the body surface (also called mesh) can be inferred using methods such as skinning methods. The number of *degrees of freedom (DoF)* of the pose is defined as the number of independent variables necessary to specify the state of the skeleton. The vector space of all possible skeleton configurations is called the *state space*. In other words, a set of independent parameters defining the positions, orientations, and rotations of all joints that produce poses forms a basis of the *state space*. Each pose can be described by a state vector $\Theta = (\theta_1, \theta_2, \ldots, \theta_N)$.

When dealing with human models, an appropriate representation must be defined depending on the intended application. The complexity and accuracy requirements may significantly impact the choice of representation. However, having a tractable model with low computational cost also needs some simplifying assumptions. Therefore, a good representation is not necessarily a complex representation but is a representation that matches the application's requirements with the minimum set of defining parameters. In the literature, the treatment for describing a pose can be split into two primary types of representations: joint positions and joint rotations. In the following, we describe them in more detail.

2.1.2.1 Representing Pose by Joint Positions

Each pose can be defined by the 3D position of the joints in a global or local coordinate system without representing the joint orientations. In the computer vision literature and for the applications such as pose estimation, pose tracking, and motion prediction, the pose is defined as the 2D joint positions in the image or video frame or 3D joint positions within the camera coordinate system.

Some of the proposed approaches for motion synthesis [3, 26, 27, 2, 1] also used local joint positions in a root coordinate system to represent the pose. This usually requires a normalizing step where the join lengths are normalized by targeting the data into a unified skeleton. One of the main advantages of representing the pose by joint positions in motion synthesis is that the joint positions can be defined independently of each other, meaning that changes (and also errors) in one joint position do not influence the position of other joints. Therefore, errors cannot propagate through the kinematic tree, causing artifacts such as foot-sliding (as is the case in representing pose

by joint rotations). However, for several reasons, such representation is not compatible with the animation and rendering engines' requirements. First, during synthesis, the generated poses are not necessarily constrained by a parameterized skeleton and limb rigidity. Therefore, additional constraints such as bone length constraints are added to the loss function during training and a corrective re-projection onto a valid character skeleton is applied to the generated samples during the motion synthesis. Second, the skeleton configuration cannot be fully represented as the rotations about the longitudinal axis of the joints is missing. Third, a re-projection onto a valid character skeleton is needed during synthesis, making it unsuitable for real-time application.

2.1.2.2 Representing Pose by Joint Rotations

To represent the pose by joint rotations, an open-chain structure, called a *kinematic tree*, represents the hierarchy of joints. This introduces a parent-child relationship among the joints where each joint has a single parent except the root which is the ancestor of all joints. Each joint is located and oriented relative to its parent coordinate system. The location of each joint in its parent coordinate system is fixed during motion and is defined as an initial offset, $\mathbf{t}_j \in \mathbb{R}^3$. Thus, each pose is defined by initial offsets, $\mathbf{t}_j$, and a sequence of joint rotations, $\mathbf{R}_j = \{\mathbf{R}_j^1, \mathbf{R}_j^2, \ldots, \mathbf{R}_j^T\}$ in their local coordinate system, where $\mathbf{R}_j^t \in \mathbf{SO(3)}$ is the rotation at frame t. Only the root's (typically pelvic) position and orientation are defined in the global coordinate system for each frame. In this way, given a predefined hierarchy, a set of predefined joint offsets, and the set of joint rotations for each frame, the character is posed in each frame. This is done through the hierarchical transformations in the recursive traversal of the kinematic tree starting from the root (called *forward kinematics* operation). Then, given the root's translation and rotation, the character body is placed and oriented in the global coordinate system. The degree of freedom in each joint (except the root which has 6 degrees of freedom) of a human can be 1 (Revolute joints), 2 (Flexion/Extension joints), or 3[1] (Ball-and-socket joints). However, in most data-driven approaches, each joint is represented by 3 or more degrees of freedom. Then, it is assumed that the model learns the lower-dimensional manifold automatically, or a dimensionality reduction operation is applied to the data before feeding it to the model.

[1]This is a simplifying assumption in animation for some of the joints such as shoulders where the links are not fully rigid. These joints are usually represented by 6 DOF in biomechanics.

There are different types of rotation parameterization to represent the joint rotations. The most common parameterizations are rotation matrices, Euler angles, axis-angle vectors, and Quaternions. The choice of parameterization depends on the performance in the application of interest, as each parameterization has its own advantages and disadvantages. One of the essential points to consider is the presence of singularities which are defined as locations in the parameter space that define the same joint orientation. These singularities may result from the choice of parameterization or some physical reality. In general, all the three-dimensional parameterizations have at least one singularity [28].

Representing the poses by joint angles is more common in the computer graphics community as it inherently follows the constraints imposed by a parameterized skeleton. In addition, representing the poses by joint angles is more compatible with the existing rendering engines to render the whole body. One of the main problems with this type of representation in the task of motion synthesis is that a small amount of error in the joints higher in the kinematic tree propagates through the descendant joints causing significant errors in the end-effectors. This usually causes artifacts such as foot-sliding. Pavllo et al. [29] proposed using joint angles to represent the pose and then performing forward kinematics at each frame during training and computing loss over joint positions. This implicitly emphasizes the joints that are higher in the kinematic tree, such as the root, which has the highest impact on the pose configuration. However, one drawback of this approach is the computational cost of the forward kinematics at each frame which slows down the training procedure. Therefore, we proposed a hierarchical loss over joint angles, which weights each joint's error based on its impact on the reconstructed pose [30]. We set the weight of each joint as the maximum path length from that joint down to all of the connected end-effectors in an average body skeleton. The evaluations were very close to [29] while our model was around 35% faster during the training. Several state-of-the-art approaches [31, 32, 5, 4] use both joint positions and rotations in the root coordinate system to incorporate the advantages of both representations.

2.1.3 Preparing Motion Sequences for Modelling Pipelines

We described the most common ways of representing human pose and motion in the previous sections. To convert the representations mentioned earlier into a format that is suitable for machine learning frameworks, some additional/optional steps are required.

2.1.3.1 Adjusting Sampling Rate

The *sampling rate* of the motion capture data is defined as the number of poses/frames per second. It is initially set by the motion capture system before capturing phase. A high sampling rate is needed to capture the detailed trajectories of the limbs in the movements with quick changes. However, all motion capture systems are limited by a maximum sampling rate. Before using the data for training and modelling, the sampling rate might be adjusted based on computational power, memory limits, and model requirements. Since most original data are usually recorded at a high sampling rate, there is usually a down-sampling preprocessing step [27, 33, 3] to reduce the size of training data and comply with the mentioned limitations.

2.1.3.2 Converting from One Representation to Another

Depending on the design and application-related requirements, there might be a need to convert the recorded representation to the representation of interest. This includes changing one type of rotation representation to another type or converting the rotation representation to the 3D position of joints and vice versa. For example, many datasets are stored in BVH file formats, representing the skeletal pose using Euler Angles. However, Euler Angles are not very common in statistical approaches due to some problems such as Gimbal Lock. Therefore they are converted to Exponential maps, quaternions, or joint positions prior to being fed to the model.

2.1.3.3 Retargetting to a Uniform Skeleton

Motion capture datasets are usually recorded from multiple actors with different body shapes and sizes. In addition, different skeletal structures may represent motion in other datasets. Therefore, data are usually retargeted to a uniform skeleton structure with the same skeletal structure, body size and proportions. One common way of retargeting is first copying the corresponding joint transformations from the source character to the target character and then doing an inverse kinematics [34]. This can also be done using available animation software packages such as Autodesk MotionBuilder [35], or Blender [36].

2.1.3.4 Making the Representation Invariant to Global Translation and Orientation

The global position and orientation of the skeleton are usually described explicitly by root position and orientation. However, the motion representation needs to be invariant to the global translation and orientation in the statistical methods. In this way, two similar motions with different global translations and orientations will be equal in the new representation space. One common approach is to define an imaginary bone by projecting the skeleton hip on the ground and defining the skeleton local to this imaginary bone transformation. Then the global motion of the character is defined by the translational (in x-y plane) and angular (about z-axis) velocities of the imaginary bone where the position and orientation are updated at each frame by integrating the velocities.

2.1.3.5 Time Warping

Some approaches pre-process similar motion samples such that they aligned in time. The criteria for this alignment can be defined by phase, foot contact, root rotation and position. The process of aligning motion samples is also called time warping [37].

2.1.3.6 Dimensionality Reduction

As it was mentioned previously, each pose is usually represented with 3 or more degrees of freedom for each joint. However, this enforces a considerable amount of redundancies as many joints have only 1 or 2 degrees of freedom. In addition, the space of natural poses (and also movements) have an intrinsic representation in a lower-dimensional space. Therefore, some approaches apply a dimensionality reduction step on individual poses or motion clips and represent data by a lower-dimensional space. The advantage of dimensionality reduction is two-fold: 1) training and synthesis are performed more efficiently with fewer computations, 2) it constraints the optimization to explore the sub-spaces which represent the natural (or statistically likely) poses or motions resulting in higher quality in motion synthesis. Principal Component Analysis (PCA), its other variations such as Kernel PCA [38] and probabilistic PCA [39], and Gaussian Process Latent Variable Models [40] are some of the dimensionality reduction approaches that have been commonly used in the context of human motion modelling [41, 42, 40].

2.1.3.7 Adding Additional Signals

In addition to the data which represent the motion kinematics, there might be other information that is either implicitly embedded in the low-level representation of the motion or originated from other sources that are associated with the motion. Depending on the application, providing these features explicitly to the model may improve the performance or add other functionalities. We can group these features into the below categories:

Control Signals: The control signals are used to control the motion and behaviour of a character by the player or animator. These inputs are provided by a game-pad or other input devices. This covers a wide range of controls such as moving trajectory [43, 29, 3, 27], action type [30, 44, 5], motion style [33, 7, 30] or interacting with the other objects [44, 5].

Environment Signals: The signals are the results of directly interacting with the environment, such as foot and hand contact [3], and altitude of the terrain [4]. These signals can usually be recovered from kinematics motion information. For example, foot contact can be detected when both the height and velocity of the toe or heel go below certain thresholds.

Other Kinematics Information: In addition to the joint angles or joint positions, other features which are implicitly presented in the motion kinematics such as joint velocity and acceleration [44, 5, 4], pace [43, 29], phase [4], joint angle constraints, and prior pose and motion space can be provided to the model. These data are usually provided to the model in two ways: 1) augmented data to the input, which acts as extra signals helping the model improve its inference, 2) used to add additional constraints to the objective function. An example is penalizing deviations from prior pose or velocity.

Associated Signals: These signals originate from other sources and cannot be extracted from motion. However, there is a loose connection between them and generated motion. One example of such signals is the agent's speech (or lip motion) which may have a relationship with the movement patterns and gesture style [33]. Another example is the "loose" environment interactions (such as the motion of other objects, weather, daylight, or map. We used "loose" to separate it from "environment signals", which can be extracted from motion kinematics).

In addition to the reasons mentioned above for adding these signals, almost all of them help with a critical problem in motion synthesis, *mean collapse*. Mean collapse is defined as regressing to

the mean pose since it is an a-priori minimizer of the minimum squared error objective. It usually happens when there is a high ambiguity in the future poses. Thus, providing these extra signals can disambiguate pose prediction.

2.1.3.8 Data Augmentation

Existing motion capture datasets are usually small and limited in the type of motions. In addition, the representation of skeleton configuration, representation of motion, data preparation, and file formats in datasets are usually different, which makes it challenging to integrate and use them in a single framework. Through data augmentation techniques, the number of training data can increase without a need for extra data. Some of the most common data augmentation techniques for motion data are:

Motion modification: mirroring the motion relative to the sagittal plane is one of the most common approaches which doubles the number of samples. Some of the works also added the reversed version of the motion sample to the dataset [27]. However, the resulting motions might not be precisely in the space of valid motions. Unlike images, there are not many options for augmenting motion data with different transformations as the motion is already normalized spatially and ego-coordinated.

Unifying different datasets: One way to address the lack of enough training data is to convert motion and skeletal representation in different datasets into a unified representation. Holden et al. [3] retargeted multiple datasets into a unified standard skeleton structure originally defined in CMU database [45]. However, this needs a different and separate retargeting script for each representation presented in the collected datasets. In another attempt, Mahmoud et al. [15] unified several major motion capture datasets by representing them in a common framework and parameterization. A method called MoSh++ [19], was used to convert mocap data into a statistical body model called SMPL [46]. SMPL model provides the skeletal representation (pose) as well as fully rigged surface mesh (shape).

Adding synthetic samples: Synthetic samples that are generated with minimal control and in an offline phase can be used in real-time motion synthesis frameworks. We [30] recently proposed a method that generates new short motion clips in many variations, which can be used for other frameworks such as motion matching [47] for long-term motion generation.

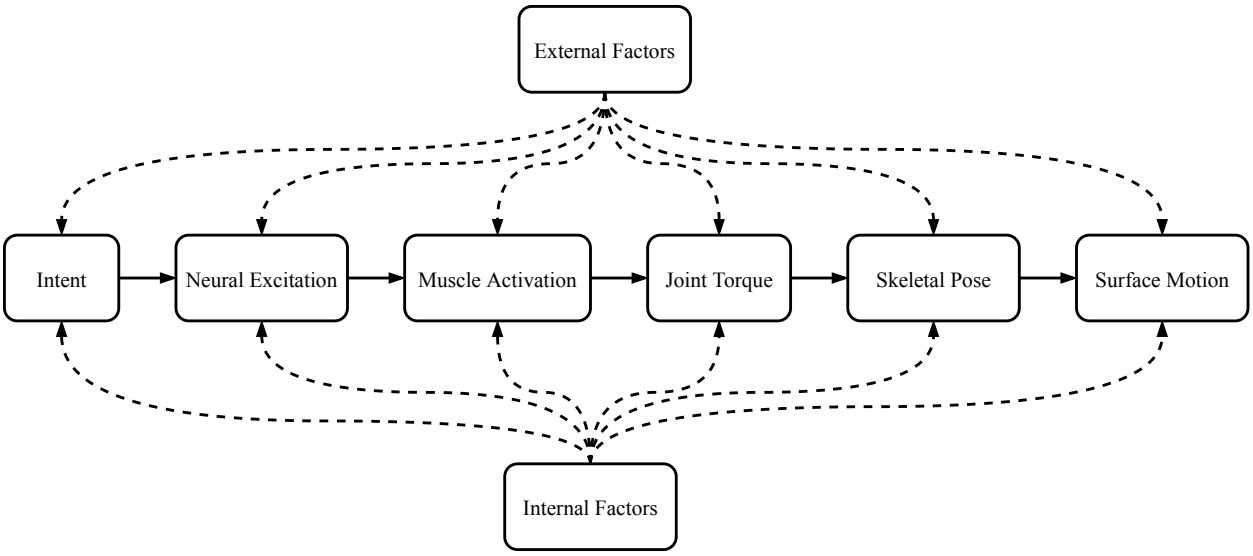


Figure 2.3: The hierarchical movement generation process.

2.2 Human Motion Modelling

Human motion can be defined as the coherent movement patterns of body parts across time. It conveys a rich source of information about intentions, action type, identity, personal characteristics, age, gender, and emotions that our visual system can perceive efficiently. On the other hand, our visual perception system can accurately, reliably, and quickly detect such biological motions and infer detailed information for an adequate interaction. Information processing involved with human motion perception is also performed in multiple stages: detection, structure from motion, action recognition, and style recognition [48].

Human movement is the result of a hierarchical generative process where the space of generated movement characteristics is shaped by a set of factors that influence the process of generating movement in different steps (Figure 2.3).

The factors that influence human movement can be categorized into two major groups: *internal factors* and *external factors*. Internal factors root in humans' ego state, such as intention, physical and neurological characteristics, style, etc. The external factors are described as the stimuli which are part of the surrounding environment, such as other agents, objects, gravity, etc. Such a high-dimensional source of influences results in a very large output space of movements. However, building such a hierarchical movement generation process that includes all of these processing

steps and can be parameterized with such a high-dimensional source of factors is impossible yet. Therefore, so-far proposed frameworks are designed to model a subset of actions, only take a subset of factors, implement a subset of steps, and output a coarse representation of human movement.

One of the common (coarse) characterization of human movement is parameterized by *content* and *style* as the two main ingredients defined in a perceptual perspective [49]. Content can be defined as the main class of motion which is initiated in the intent step and is not influenced by other factors. On the other hand, style is defined as the different variations in the same class parameterized by internal and external factors. For example, walking is identified as content, while different variations of walking such as masculine-vs-feminine and sad-vs-happy are identified as style. Such characterization in character animation frameworks is adapted as generating the content and modulating it by style as an additional control parameter or transferred from other sample movements.

In a very general form, the human motion model can be described as a tool for representing human movement, which is more suitable than raw representations for tasks such as analysis, recognition, or synthesis. Such models are prevalent in many areas, such as computer animation, robotics, biomechanics, and computer vision. In computer animation, such models play a crucial role in generating new and controllable character animations, key-framing, and preprocessing motion capture data. In robotics, such models can be translated to the motor controller of physical robots or can be used to control the machines in interaction with unknown environments. In biomechanics, these models provide clinically meaningful analysis of people's movement. In computer vision, such models can provide priors or constraints in tasks such as pose estimation or action recognition from video or image data to narrow down the space of true poses and motion sequences.

A character animation model is defined as a framework that creates *motion sequences* from a *motion space*. Each element in the motion space is defined as a fixed-length or non-fixed-length continuous *motion sequence* which corresponds to a set of *model parameter* values $\boldsymbol{\theta} \in \mathcal{C}$ where $\mathcal{C}$ is the space of model parameters. As we discussed earlier, the space of motions that humans can perform has too many degrees of freedom and building a model with a motion space covering such endless variety is not feasible at the moment. Thus, designing a motion model aims to cover a subspace of motion space with a limited number of factors to describe the generated motions.

There are different requirements for a character animation framework that should be satisfied depending on the application:

- Controllability: In most scenarios, the characteristics of the generated character movements need to be controlled by a user. In fact, one of the main advantages of utilizing a motion model over just using recorded data is the ability to control the virtual character and manipulate its motion and expressive attributes by high-level descriptors. For example, in video games, a wide range of motions should be generated and controlled by the gamepad inputs. In animated films, the virtual characters should be directly and intuitively controlled by the animators in an animation interface

- Implementability: The framework and the underlying modelling ideas need to be efficiently implemented given the limitations in the amount of data, hardware, software, time, and manual work

- Interpretability: Interpretability is an essential requirement for applications in which the motion is analyzed. Examples of such applications are gait and biomechanical analysis. However, this is significantly helpful to have an interpretable model (or representations) to improve the framework's design in other applications such as character animation.

- Believability (Naturalness): the human visual system is exceptionally sensitive to detecting and retrieving expressive details and subtle information from peers' movements. Therefore, even minimal deviations from a natural movement are detected by our visual system. This raises a need to generate motions as natural and convincing as possible and beyond the uncanny valley. This includes *quality*, *diversity*, *consistency*, and *physical realism*. These are four main components that play an essential role in achieving a high level of realism:

    1. Quality: The created motion samples should lie on the manifold of natural motions.
    2. Stochastic Characteristics: Human motion is stochastic in nature with high uncertainty in the future movement trajectories. Our body components never traverse the same path given the same aim, intention, and planning. This stochastic nature and variability are some of the fundamental characteristics of biological motion. Therefore, one of the main challenges in generating realistic character animation is to avoid generating duplicates,

which causes the perception of the model to be more mechanical rather than natural, and to generate many variations of the same movement. This is one of the common problems in models such as motion graphs [50], motion matching [17], and deterministic statistical models [2, 1, 26, 3, 4] as in long-term the repeated patterns of movement are easily detected by our visual perception system.

3. Consistency: The generated movement should be consistent with the desired holistic attributes (such as age, gender, and style) and also control signals

4. Physical Realism: The movements generated by a model are expected to follow the physics rules and constraints in the involved biomechanics and interaction with the environment. Our visual system implicitly learns the relationship between movements and the laws of physics. Thus some of these laws should be obeyed in the generated animations. One of the main challenges in kinematics-based and data-driven approaches is that no prior physical assumption is integrated into the model, and obeying such physical laws is not guaranteed. One common issue in generating character animation using machine learning approaches is foot-sliding, as the friction laws have not been formulated into the framework.

- Low latency: Real-time interactive systems such as video games require models with the lowest possible latency. Given a set of control parameters in such systems, a real-time generation of the content, including the corresponding character movement and according to the required frame-rate, is desired. Having low latency is one of the main challenges considering memory and computational power limitations.

- The fewer data, the better: Many kinematic approaches rely on pre-recorded motion capture datasets. However, recording high-quality datasets suitable for such models is a tedious and long process, including much manual work. One of the potential improvements toward using such approaches is to minimize the amount of needed data

- Versatility: In most character animation scenarios, we may need the model to generate multiple actions with different attributes and styles. The model should be able to cover the space of all needed motion variations.

There has been an extensive amount of work over the years dedicated to the design and development of new models with the aim of producing character animation with minimal manual work in the video game and animated movie industries. Our primary focus in the review of different models is on machine learning and deep learning approaches for modelling human motion. These models are considered a subset of statistical models by which the statistical variations of a training dataset are learned to generate new samples. These data-driven models have shown great potential in modelling human motion and character animation, which was inspired by the considerable success in other fields such as time-series modelling, image and video synthesis. One important note is that the range of the motions that such models can generate heavily relies on the subspace of the motion samples provided in the training dataset. In general, statistical models cannot extrapolate significantly out of the sub-manifold from which the training data is sampled. This sub-manifold is defined by many factors such as action type, intra-class variations, and the human actors performing the samples. Another critical consideration for such models is the size of training data. Such models usually rely on a large number of samples in order to be able to cover the desired sub-space of the motion space effectively. However, large datasets often pose other challenges such as long training time and mean collapse problems. In the following, we describe the most common techniques for modelling human motion in more detail.

2.2.1 Deterministic Models

We first provide an overview of deterministic models. These models are constructed based on the assumption of one-to-one mapping between the input and output.

2.2.1.1 Dimensionality Reduction

Dimensionality reduction techniques are usually used to transform high-dimensional data into a lower-dimensional space which is useful for reducing the redundancies presented in data, reducing the memory storage usage, requiring less computational power, and increasing the training speed. In addition, since motion data lies on this lower-dimensional manifold, the model is less prone to explore the space of unnatural and statistically unlikely poses and motions during optimization and synthesis.

One of the earliest works which exploited dimensionality reduction in modelling human motion

was done by Bowden [51]. First, PCA was applied to the pose data to discard some of the redundancies and reduce the dimensionality of the pose representation. Then a fuzzy k-means clustering was applied to segregate the dataset into its composite clusters. Each cluster then underwent another PCA upon its members as a local dimensionality reduction. Applying a dimensionality reduction locally on each subspace acted as a non-linear dimensionality reduction on the global high-dimensional manifold. Finally, a first-order Markov chain was used to learn the dynamic transitions across the temporal aspect of motion. Similarly, Chai and Hodgins [52] exploited the idea of using local PCA on the clusters of k-nearest-neighbour poses to infer the full-body skeleton given an ill-defined pose (a sparse set of joints). This way, the optimization was performed in a low-dimensional space for each ill-defined pose query by minimizing some energy terms. In the following work, [53], they proposed a constraint-based motion synthesis by statistical modelling of human motion in a low-dimensional space which enforces generating natural motion that follows user-defined goals. First, the dimensionality of the joint angle representation was reduced by applying PCA to each pose. Then, in a probabilistic step, the new motions were synthesized in a constraint-based maximum a posterior (MAP) framework defined in the low dimensional latent space. In the MAP framework, the most likely motion given the user-defined constraint was defined as a combination of motion prior and a likelihood term. The motion prior was represented by a linear time-invariant state-space model that describes motion's dynamic behaviour in the latent space. The likelihood term, which measures how well the generated motion matches the constraints, was defined based on the joint positions or joint angles distances. Finally, the optimization was performed using sequential quadratic programming (SQP) [54].

Using PCA in the context of motion data is not limited to the pose space. Tailemanne et al. [55] used PCA to reduce the dimensionality of walk cycles and to model the different walking styles. The training data were first preprocessed by resampling all walking cycles to a fixed-length (40 frames) using SLERP [56]. The joint angles represented by the quaternions were then converted to exponential maps as the space of quaternions is non-linear and cannot be reduced by PCA. Then a PCA was applied on the preprocessed walking cycles, and 23 first principle components were kept to retain 90% of the cumulated percentage of information. Each principal component of each style was then modelled as a Gaussian process. Each walking cycle was sampled during motion synthesis from the corresponding principal components of the style of interest. Each generated walking

cycle was also resampled based on a learned Gaussian distribution upon each style's duration of walking cycles. Finally, all the generated walking cycles were concatenated to construct the entire motion. Modelling walking cycles and their duration using the Gaussian model introduced some randomness in the synthesized sequences. Troje [57] transformed the periodic movement of the joint in locomotion data into the frequency domain by Fourier transform resulting in a fixed-length representation vector. Since the number of dimensions in this Fourier-based representation were still higher than the number of samples (walkers), they applied PCA on the resulting representation vector to reduce the dimensionality. Having a lower number of dimensions compared to the number of samples allowed them to construct linear classifiers on gender and different emotions with the ability to generalize to new locomotion patterns.

Other dimensionality reduction approaches have also been extensively used in the context of motion modelling. Qu et al. [58] proposed reducing the dimensionality through Isometric feature mapping (Isomap), a non-linear dimensionality reduction approach that is designed to preserve geodesic distances between the samples. This is done by constructing a graph whose nodes are data samples and edges are defined based on the geodesic distance between the nodes. After reducing the dimensionality of the pose space, they segmented motion data by fitting linear dynamic models to each segment. New motion sequences were then generated by concatenating generated segments where the intermediate frames were synthesized by solving a linear equation in a second-order Markovian model.

It is also possible to reduce the dimensionality of a movement across temporal dimension (over trajectories) rather than spatial dimension (over poses). Therefore, the data to be reduced in dimension is represented as functional data (trajectories) rather than multivariate data (poses). Conventional PCA is used to investigate the modes of variations in multivariate data but cannot be applied to functional data. Functional principal component analysis (FPCA) is a statistical method for extracting the dominant variations of functional data. Unlike PCA, the principal components in FPCA are represented by a set of orthonormal *eigenfunctions* of the autocovariance operator. Samadani et al. [59] used FPCA to extract a set of features expressed as FPCA coefficient of movement time series data. They first aligned similar movements and converted them to fixed-length sequences using a piece-wise linear resampling. Then, the time series features were decomposed into a temporal and weighted linear combination of a set of basis functions using

basis function expansion (BFE) [60]. The BFE representation was then transformed into a low-dimensional space using FPCA. The resulting low-dimensional feature transformation was used for the classification and generation of hand movements labelled by holistic attributes.

Min and Chai [61] used Functional PCA to construct a morphable function for compactly representing the style variations within the same type of motion primitive. First, each motion sequence was decomposed into distinctive motion segments, which were automatically placed into a cluster belonging to the same motion primitive. Then the segments inside each cluster were aligned using the dynamic time warping technique. A functional PCA was then applied to all warped motion segments associated with each motion primitive cluster. A morphable function was constructed over low dimensional representations to represent the style variations in each motion primitive cluster compactly. Next, the morphable motion primitives were organized into a finite directed graph where each node represents one morphable motion primitive, and each edge represents the probability of transition between each two motion primitives. The basic structure of their graph was similar to the motion graphs [50], with the difference that rather than storing recorded motion segments at each node, it is probabilistically generated in real-time. The transition distribution functions between the nodes were modelled using Gaussian Process (GP). They also annotated each motion primitive with the surrounding contact information and semantic knowledge to be used during the generation process as user-defined control constraints. The key idea in this work was decoupling the finite structural variations from continuous style variations presented in human motions. The generation process was also performed in two steps based on this decoupling modelling. First, a sequence of morphable motion primitives was synthesized by concatenating morphable nodes via a graph walk. Then, the movement segments corresponding to each node were generated via a probabilistic sampling, generating a wide range of style variations in each run. To control the motion, a Maximum A Posteriori (MAP) framework was designed where the optimal node at each graph step was found using a gradient-based optimizer by maximizing the posterior probability defined by the transition, contact-awareness, and control terms.

2.2.1.2 Recurrent Neural Networks

Most of the classical approaches such as PCA-based approaches (see Section 2.2.1.1), Boltzmann machines (see Section 2.2.2.2), or Hidden Markov models (see Section 2.2.2.3) have high compu-

tational complexity or are unable to efficiently model the non-linear dynamics of human motion, especially for long-term multi-modal motion synthesis. Recurrent Neural Networks (RNN) are among parametric models in which the run-time computational complexity is independent of training data size ($O(1)$). These models are capable of modelling non-linear dependencies with adjustable complexity and generalizability. The recent advances in training these models such as parallel computing, novel architectures with long-term state memory [62, 63], addressing vanishing/exploding gradients problem, and attention mechanisms [64] have attracted more researchers to exploit such architectures for modelling human motion.

Recurrent Neural Networks are a class of autoregressive models that summarize all previous states into a fixed-length vector called *hidden state* acting as a memory. RNNs are designed as directed networks that can be trained using a gradient-based optimizer.

A notable approach that applied RNNs to learn human motion was introduced by Fragkiadki et al. [1] who proposed a hierarchical *Encoder-Recurrent-Decoder* (ERD) architecture which encodes each pose into a feature vector where it is recursively processed through a two-layer LSTM [65] network (Figure 2.4). Augmenting a typical RNN model with non-linear encoder and decoder networks before and after recurrent layer transforms pose data to an intermediate representation which is more efficient and expressive to learn the long-term dynamics of human motion. Encoder and decoder networks were implemented using fully connected neural networks and while the recurrent network was implemented by two layers of LSTMs. It is often common in recurrent neural networks that during training, the ground truth input is fed to the model at each time step. Therefore, the model cannot learn to recover from its own mistakes. During motion synthesis, the prediction is fed back to the model in the following time step, which causes the accumulation of small errors at each time step (called *exposure bias*). To address this problem, the input to the ERD was corrupted by Gaussian noise with progressively increasing standard deviation as a type of *curriculum learning*. However, it is usually hard to tune the amount of noise.

Motivated by machine translation strategies [66], Martinez et al. [2] proposed a sequence-to-sequence architecture to model human motion, including an encoder that summarizes initial frames into an internal representation and a decoder that takes the internal state as an input and predicts future motion based on the maximum likelihood estimate (Figure 2.5). Their proposed encoder and decoder shared their weights to accelerate the convergence. To address the problem of exposure

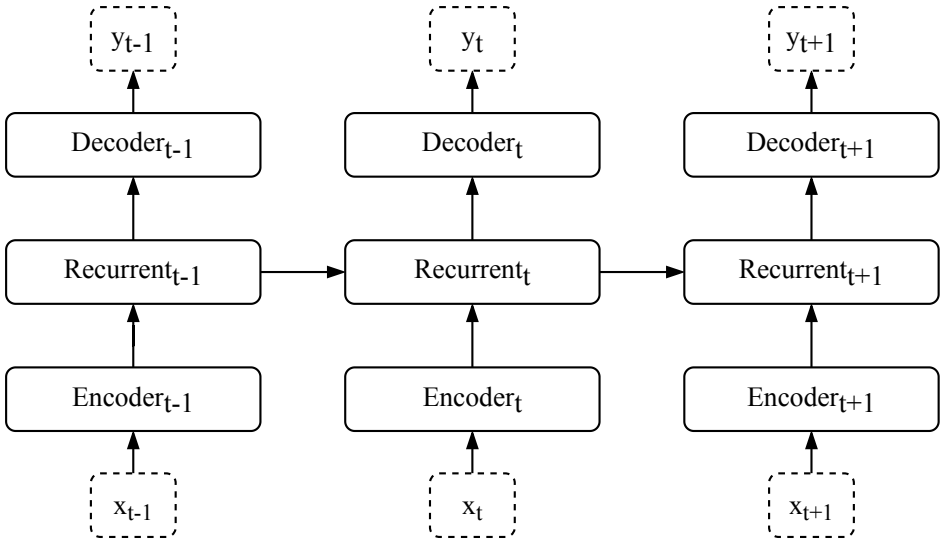


Figure 2.4: The Encoder-Recurrent-Decoder Architecture proposed by Fragkiadki et al. [1]. The pose space is transformed into an intermediate representation space using *Encoder* where the dynamics are learned by *Recurrent* network. Then, the output is reconstructed by transcribing the output of Recurrent network to the pose space by *Decoder* network

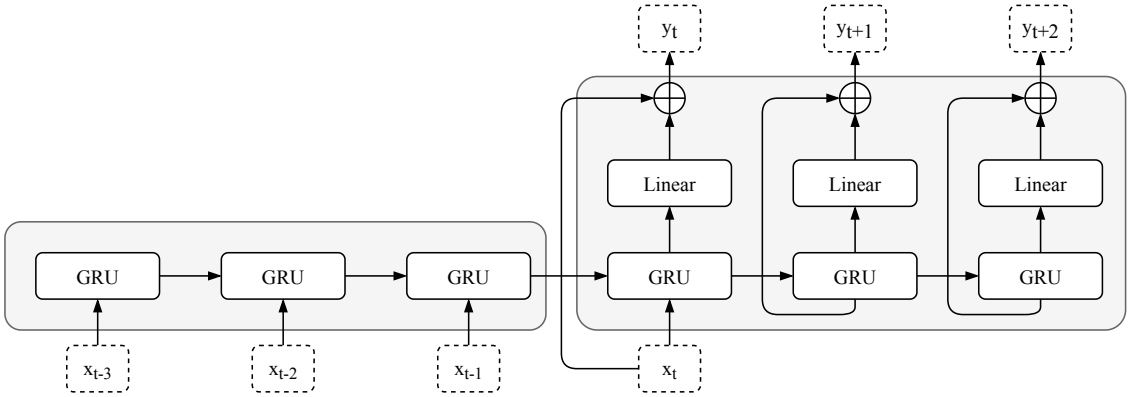


Figure 2.5: Seq-2-Seq Architecture proposed by Martinez et al. [2]. The ground truth initial sequence is fed to the *encoder*, and joint angle velocities of predicted sequence is computed by the *decoder* and linear networks that takes its own sample as input during training.

bias, the decoder at each step was fed by its own predicted samples at previous steps during the training phase. One benefit of this approach is that unlike injecting noise in a curriculum learning scheme [1] where the amount of noise should be tuned at each time step, no hyperparameter tuning is needed. By adjusting the length of generated sequence during training, this strategy also helps the network to produce plausible long-term motions. Another proposed strategy was to enforce the model to learn joint angle velocities instead of absolute values via residual connections in the decoder to address the problem of the discontinuity between the initial sequence and prediction

sequence in the seq-to-seq models.

The approaches which represent joint angles by Euler Angles or Exponential Maps (Axis-Angles) suffer from non-uniqueness, discontinuity in the representations space, singularity, and high computational complexity of composing transformations. To address these problems, Pavllo et al. [43] used quaternions to represent joint angles that are free of discontinuity and singularity, computationally more efficient, and numerically more stable. The proposed model was based on a deterministic recurrent neural network for short-term prediction and long-term locomotion synthesis. They used a two-layer Gated Recurrent Unit (GRU) with learnt initial state from the data for both tasks. For the task of long-term locomotion synthesis, the input was augmented with translations and control signals. To enforce the generated angles to represent valid rotations, they added a normalizing layer that normalizes the quaternions to have unit length. For short-term prediction, they used residual connections to predict angular velocities inspired by [2] which shows improvements in addressing the discontinuity in motion prediction when using absolute values.

Most proposed approaches model human motion without an explicit representation of the kinematics tree, while this structural information can be potentially very useful for modelling the movements. By explicitly modelling spatiotemporal interactions between joints, Jain et al. [67] combined the explicit representational power of spatiotemporal graphs with the implicit sequential learning of RNNs. The proposed model was designed by extending *spatiotemporal graphs* (st-graph) into a feed-forward mixture of RNNs called *Structural-RNN* (S-RNN). In S-RNN each component factor (including spatial, temporal, and spatiotemporal factors) was represented using an RNN. The component factors with similar semantic functions were allowed to share the same RNN weights, keeping the overall network scalable without losing any learning capacity. Aksan et al. [68] demonstrated an alternative method of incorporating structural information, proposing a *Structured Prediction Layer* (SPL) where the prediction of each joint at time t was conditioned on the joint's previous state and the current state of the parent joint. Therefore, at each time step, the joint angles are predicted starting from the root to the leaf nodes in the kinematic tree. They integrated the proposed layer in a variety of baseline architectures and showed improvements for the task of motion prediction. Note that here we summarized some of the deterministic models which use RNN to model the spatiotemporal dependencies. However, these architectures can be used as building blocks of probabilistic models as well.

2.2.1.3 Convolutional Neural Networks

Convolutional Neural Networks have also been exploited for modelling human motion. Unlike RNNs and autoregressive models, these models are not suitable for real-time motion synthesis. However, they have shown potential for post-processing or manipulating the existing motion sequences.

Holden et al. [26] proposed a model based on Deep Convolutional Autoencoders [69] composed by an encoder and decoder with convolutional shared weights. The model was trained in an unsupervised manner on a large human motion dataset to learn the underlying manifold of the fixed-length sequences of motion. The learned manifold was then used for various applications such as filling in missing data, motion interpolation, and motion comparison.

In the following work, Holden et al. [3], built upon previous work utilizing the convolution autoencoder where synthesis and editing of motion data were performed in the space of the learned motion manifold. The approach is similar to GPLVM motion models [70, 71] (see Section 2.2.2.1 for more detail) since manipulating existing motions or synthesizing new motions is in a low-dimensional non-linear motion manifold which represents the space of plausible movements and poses. First, a bi-directional convolutional autoencoder (with shared convolutional weights in encoder and decoder) was trained on a large motion database. Then, on top of the decoder network of the trained autoencoder, another convolutional feed-forward neural network was stacked which maps the high-level user control inputs such as trajectory to the hidden space motion representation of the autoencoder. To disambiguate the locomotion and provide a stronger control signal, the motion trajectory was augmented with the timing of the foot contacts. To this end, another convolutional neural network was pre-trained to automatically predict the foot contacts given the trajectory curve parameters (translational and rotational velocity). The outline of the architecture is illustrated in Figure 2.6. Having the trained framework, existing or synthesized motions can be edited in the space of the learned motion manifold where the values of hidden units are optimized using a gradient-based optimizer given some constraints as cost functions. One explored application was motivated by style transfer proposed by Gatys et al. [72] which was originally applied on images. To transfer the style of a style motion to a content motion, the content motion is optimized in the motion manifold such that the *Gram Matrix* of the hidden units are close to the one extracted from style motion. Gram Matrix of hidden activation describes the spatiotemporal texture as a

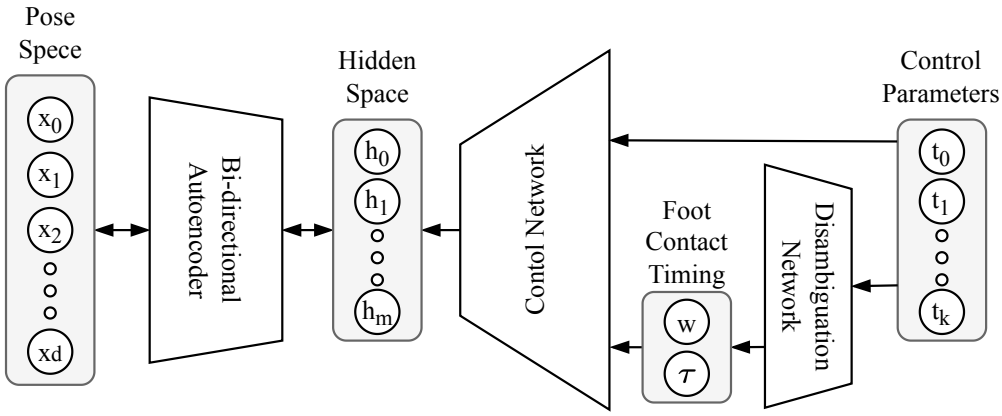


Figure 2.6: The architecture proposed by Holden et al. [3]. Editing and synthesizig motions is performed in a learned motion manifold provided by a pre-trained bi-directional autoencoder. The control network maps high-level control parameters to the corresponding motion represented in the learned manifold

representative of the motion style.

Aberman et al. [73] proposed a style transfer approach motivated by another style transfer technique in images called *Adaptive Instance Normalization* (AdaIN), which aligns second-order feature statistics of content feature in the instance normalization layer with those of style features in real-time. The proposed architecture for motion style transfer consisted of two 1D convolutional encoders for style and content motion sequences which extract a low dimensional intermediate latent code for each. The style motion sequence could be represented by 3D mocap data or 2D motion extracted from video data. The encoder for content motion is integrated with instance normalization layers to strip its style off. The content latent code is then decoded using a convolutional decoder where the mean and variance of its instance normalization layers are modulated by the style latent code using an MLP. The generated motion is then fed to a discriminator, which is trained to discriminate the generated style from the true style. The proposed approach can learn from unpaired databases of motions with style labels.

2.2.1.4 Memoryless Autoregressive Networks

RNNs address the high uncertainty in predicting the next state by having an infinite dynamic response through their hidden state as a key ingredient that summarizes the history of all previous states. Therefore, given only the current state, the hidden state helps to disambiguate the next state by providing implicit pieces of information such as timing, phase, or history of previous actions.

However, encoding a large amount of information with details in a limited hidden vector is impossible. Using more advanced RNN networks such as LSTMs and GRUs, which are explicitly designed to avoid the long-term dependency problem, might alleviate the problem to some extent. However, the latency of these models is relatively high for real-time applications such as interactive video games, where low latency is a critical requirement. This is usually due to the high computational complexity of having multiple gating networks to produce the output and update the hidden state (and cell state in LSTM). CNNs are, in general, faster than RNNs. The long-term dependencies in CNNs can be modelled by having large receptive fields across temporal direction [29]. However, the length of receptive fields in such models is limited by the amount of computational complexity. In addition, some of the approaches which were proposed based on CNNs are not suitable for real-time application as the data is not processed autoregressively.

To address the above-mentioned problems, some frameworks are designed based on a lightweight autoregressive core model with a finite dynamic response (only one or a few previous states are provided at each time step) and no memory state. To compensate for missing a memory state, additional strong control signals such as foot-contact timing [3], pace [43, 29], phase [4] can be provided to the model to disambiguate predicting the next state. Another strategy is to divide the space of training data into subspaces and implicitly or explicitly train a network for each subspace [4, 5, 44, 74].

Holden et al. [4] proposed *Phase-Functioned Neural Network* (PFNN) for a real-time character control where the *phase* (a variable which represents the timing of locomotion cycle) is explicitly provided to the model to disambiguate the next pose. In PFNN, the weights of the main autoregressive network are computed dynamically using a cyclic function called *phase function* which takes phase as input. The autoregressive network takes as input the control signals, semantic labels, geometry of the surrounding environment, and the previous state (local joint positions and velocity), and then generates the next state, in addition to change in phase (to update the phase). Conditioning the network weights on the locomotion phase helps the model to avoid mixing data from different phases without a need for the long history of past states or an infinite dynamic state such as RNNs. In their framework, each state is generated given only the previous state and a limited number of control signals in the past and future frames. This also helps the architecture to be fast and compact, suitable for real-time scenarios. The phase function was designed as a

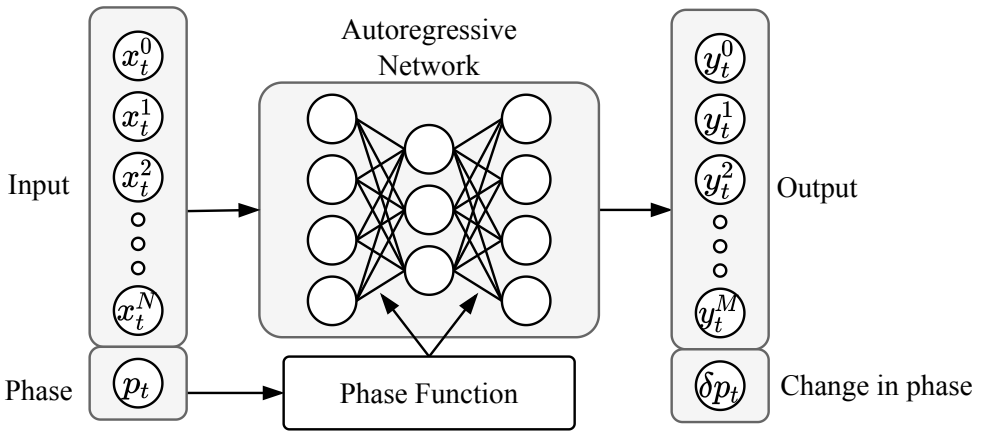


Figure 2.7: The architecture of Phase-Functioned Neural Network proposed by Holden et al. [4]. The phase function takes phase of the locomotion as input and modulates the weights of the autoregressive network. Providing the phase explicitely to the model helps it to disambiguate the next pose.

cyclic Catmull-Rom spline where the number of parameters was proportional to the number of control points. Training data was recorded as long sequences of locomotion in a non-flat studio, including different obstacles, ramps, and platforms. The simulated terrains were then generated in an offline stage by fitting a database of terrain heightmaps recorded from video games and other virtual environments. In the evaluations, PFNN showed a better performance compared to standard fully-connected networks where the phase was concatenated to the other inputs (Figure 2.7).

Zhang et al. [5] proposed a character control framework for quadruped motion control using a neural network architecture called *Mode-Adaptive Neural Networks* (MANN). The proposed architecture were composed of a motion *prediction network* and a *gating network* (Figure 2.8). The motion prediction network is the main autoregressive network responsible for predicting the character state of the next frame given the state of the current frame and user control signals based on a first-order Markovian assumption. At each time frame, the parameters of the prediction network are updated by blending a set of learnable *expert* weights where the blending weights (mixing proportions) are determined by the gating network. Similar to *Mixture of Exports* (MoE) [75, 76] each expert is specialized in a particular regime or a particular type of movement in the space of training data assigned by the gating network. Unlike Mixture of Experts, where the outputs of experts are blended, in the mode-adaptive neural networks, the weights of the experts are rather blended. This

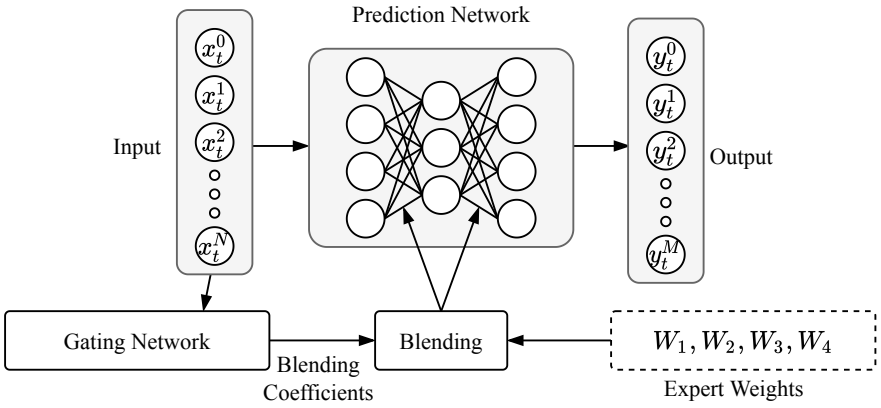


Figure 2.8: The Mode-Adaptive Neural Network architecture proposed by Zhang et al. [5] composed by Prediction Network and Gating Network. Gating Network receives the current state as input and modulates the weights of the prediction network by blending the learnable expert weights. Each expert is specialized for a subdomain in the training data space.

strategy provides the flexibility to automatically learn a variety of periodic and non-periodic action types in an unsupervised manner where no labels are required by the mode. However, MANNs need n-times more parameters plus a gating network to cover n different action types compared to ordinary Autoregressive models.

Starke et al. [44] extended the Mode-Adaptive Neural Network [5] to a model called *Neural State Machine* (NSM) for goal-driven actions and environment interactions. The base architecture of their proposed model is similar to MANN composed of prediction and gating networks. In addition to the locomotion phase, the character motion data were augmented with 3D volumetric scene geometry and goal location for goal-driven synthesis. They also used a bi-directional control scheme where the ego-centric and goal-centric inferences were mixed. In this way, a feedback signal from the goal point of view will be sent to the Neural State Machine to increase the precision accuracy of character motion towards the goal and decrease the accumulation error.

In the following work, Stark et al. [32] proposed a model for interactively controlling the players in the basketball games based on a mixture-of-experts architecture similar to their previous models with two improvements to enable their model to learn quick ball maneuvers or the interaction movements between the player and ball, other players, and the environment from motion capture database. First, they trained their model with local motion phases where the contact between some of the body parts (hands and feet) and external objects such as the ball and other parts of the environment is modelled by fitting a sinusoidal function. This helps the model to learn

fast interactions between different body parts and objects or the environment, which is usually asynchronous. Second, they integrated their model with another network that takes raw high-level control signals as input and generates compatible sharper control signals for the main model resulting in generating sharp non-deterministic movements. They used an encoder-decoder network and trained that via an adversarial training scheme by integrating a discriminator into the output of their network. In the run-time, some noise is added to the latent code to generate wide variations of movements given the same control input in a non-deterministic fashion.

2.2.2 Probabilistic Models

Most deterministic models suffer from two main problems. First, these models usually regress to the mean pose. This is because such models implicitly assume a single true next pose at each step, while, in fact, there is a high level of intrinsic uncertainty in the human motion, which grows with time. Therefore, the predicted output converges to the mean pose to maximize the likelihood. This can be alleviated by providing strong control signals. However, extracting strong control signals from training data and providing them to the framework has its own challenges. Second, given a set of control parameters, only a single, fixed sequence is generated in deterministic models. On the other hand, human motion is stochastic in nature - given the same intention and target, the joints always travel different paths. This feature usually results in repetitive and non-engaging character animation. On the other hand, the generative models model the distribution of the data so that they can be efficiently sampled. Incorporating the uncertainty which exists in data into the model can potentially overcome the limitations of existing deterministic methods.

A generative model refers to a model that learns to represent an estimate of data distribution p_{data} given training samples from this distribution. The estimated model is called model distribution p_{model} which may allow to 1) compute $p_{\text{model}}(x)$ for an arbitrary value of x, and 2) sample from $p_{\text{model}}(x)$. These models can be split into two main groups: 1) **Explicit Density** models such as Variational Autoencoder and Tractable Density models, which represent the model density explicitly and allow for (exact or approximate) likelihood evaluation and 2) **Implicit Density** models such as GANs which only provide generated samples from model density. In the following, we briefly describe the main classes of the likelihood-based models designed based on deep neural networks that have achieved state-of-the-art results in various contexts. Likelihood-based gener-

ative models are represented based on **function approximation** where the function parameter set $\boldsymbol{\theta}$ is computed so that $p_{\boldsymbol{\theta}} \approx p_{data}$. The fitting distributions procedure in general can be defined as a search problem over parameters $\boldsymbol{\theta}$ to minimize an objective function for a given data $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ sampled from true distribution p_{data} :

$$\arg\min_{\boldsymbol{\theta}} \text{loss}\left(\boldsymbol{\theta}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\right). \tag{2.1}$$

The most common objective for estimating distribution is **negative (log) likelihood** (NLL) where

$$\arg\min_{\boldsymbol{\theta}} \text{loss}\left(\boldsymbol{\theta}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\right) = \frac{1}{n}\sum_{i=1}^{n} -\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right), \tag{2.2}$$

where the parameters are found to maximize the probability of the training data [2]. This is equivalent to minimizing the KL divergence between p_{data} and $\hat{p}_{\boldsymbol{\theta}}$:

$$\text{KL}\left(\hat{p}_{\text{data}} \| p_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}\left[-\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] - H\left(\hat{p}_{\text{data}}\right), \tag{2.3}$$

where

$$\hat{p}_{\text{data}}\left(\mathbf{x}\right) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\left[\mathbf{x} = \mathbf{x}^{(i)}\right]. \tag{2.4}$$

In the following, we will provide a review of probabilistic approaches which have been exploited for modelling human motion.

2.2.2.1 Gaussian Process Latent Variable Models

Gaussian Process Latent Variable Model (GPLVM) [77] was one of the earliest probabilistic techniques to model human motion data due to its ability to model non-linear dependencies in the data probabilistically. GPLVM is an extension of Dual Probabilistic PCA [77] where instead of placing a prior on the parametric mapping's parameters, a Gaussian process prior was directly placed on the mapping function. Then, this Gaussian process prior allows for non-linear mappings by setting a non-linear kernel. Grochow et al. [70] constructed the likelihood of poses using a GPLVM. To

[2] We use log-likelihood instead of likelihood to avoid the under-flowing problem and make the optimization more efficient. Note that the log function is a monotonically increasing function

assign different weights to different dimensions based on the influence of each dimension on the final pose, a separate scaling was estimated for each dimension. This extended version was called Scaled Gaussian Process Latent Variable Model (SGPLVM). To generate new poses, an objective function describing the likelihood of new poses with respect to some user-defined constraints was optimized. This idea was used for various applications such as interactive character posing, trajectory keyframing, style interpolation, real-time motion capture with missing markers, and posing from 2D images where a specific set of constraints were defined for each application.

Levine et al. [71] augmented a GPLVM with novel connectivity prior to ensuring rich interactions between motion samples in the latent space similar to motion graphs. In the first training stage, the motion space was learned using the proposed GPLVM. Then, two separate GPLVMs were trained, one for joint angle values and another one for joint angle velocities. In the second training stage, a near-optimal control policy based on Markov Decision Process (MDP) and approximate dynamic programming were computed. During the runtime, the trained controller proposed transitions in the learnt continuous latent space in response to the user-defined tasks. Using this novel variant of GPLVM allows the policy controller to explore and propose new transitions (and therefore motion sequences) that are not presented in the training dataset while still looking natural and convincing.

Wang et al. [78] proposed a variant of GPLVM called *Gaussian Process Dynamics Model* (GPDM), which comprises modelling temporal dynamics in a low-dimensional latent space and a mapping from temporally structured latent space and the observation space. The latent-observation mapping was modelled using SGPLVM similar to [70] with an RBF kernel. The dynamics over latent space were modelled by a first-order Markov assumption where each step is defined as a Gaussian Process regression. Extending GPDM to model higher-order Markov sequences can help with modelling higher-order dependencies such as velocity and acceleration, which is more appropriate for smoothly changing dynamic sequences.

In general, although kernel-based approaches such as GPLVM and GPDM can model the non-linear and complex dependencies in the motion data, they suffer from high memory cost for storing the covariance matrix and also high computational complexity for computing the inversion of the covariance matrix, which is of the cubic order of the number of training samples.

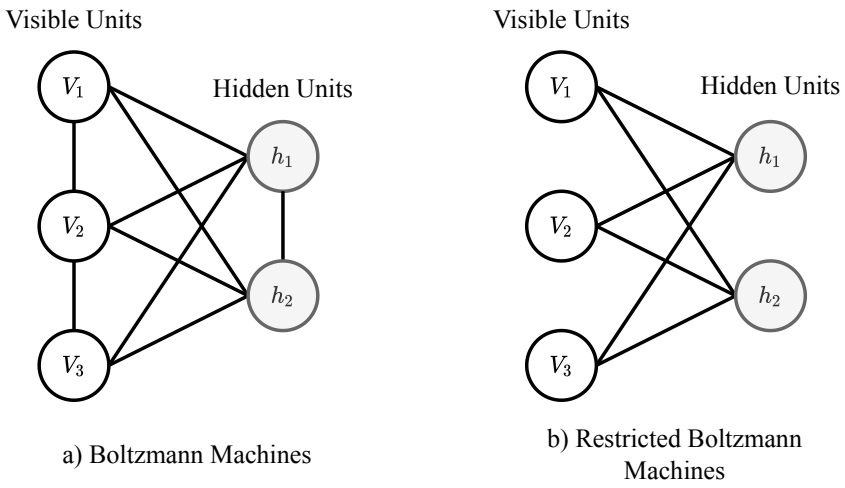
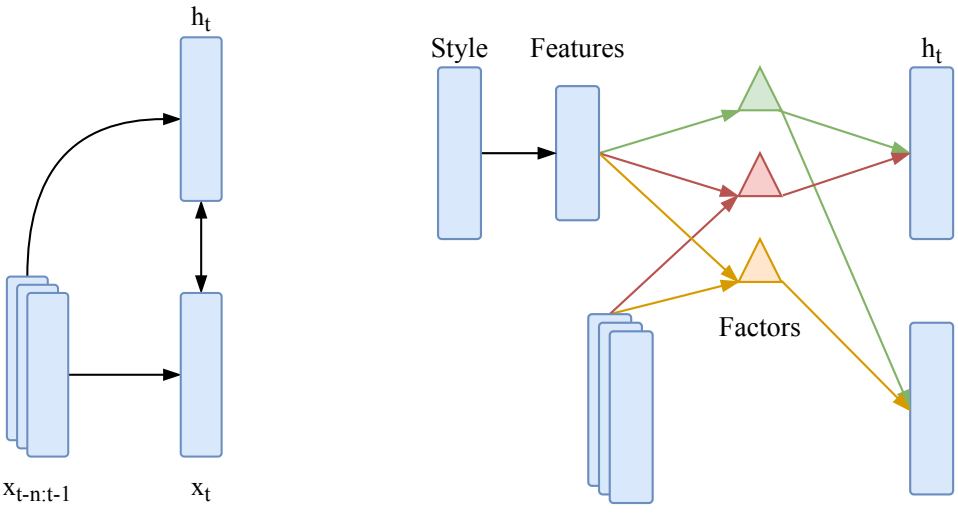


Figure 2.9: Graphical representation of Boltzmann Machines and Restricted Boltzmann Machines.

2.2.2.2 Boltzmann Machines

Boltzmann Machines [79] are a type of stochastic neural network composed of a set of observed units and a set of hidden units that are connected by undirected edges. The joint probability distribution in Boltzmann Machines is defined using an energy function which is usually learnt based on maximum likelihood. Restricted Boltzmann Machines (RBM) [80] are a class of Boltzmann Machines that does not allow intralayer connections between hidden units and visible units, which provides an efficient learning process. Figure 2.9 shows the graphical representation of Boltzmann Machines and Restricted Boltzmann Machines.

One of the earliest attempts to overcome the limitations of the traditional approaches, Taylor et al. [6], approached human motion modelling using a variation of *Restricted Boltzmann Machines* (RBM) called conditional Restricted Boltzmann Machines (cRBM) as an energy-based model where the temporal dependencies are modelled by conditioning the latent and visible variables at each time-step to the visible variables at the last few steps using directed connections. The directed connections are grouped into autoregressive connections, which connect previous n time-steps of visible variables to the current visible state and the connections which link the previous m time-steps of visible variables to the hidden state (Figure 2.10a). The former models the local and linear temporal dependencies, while the later models the global non-linear dynamics. In the cRBM, real-valued visible variables represent joint angles, and binary latent variables summarize the past



(a) cRBM architecture

(b) Factored CRBM

Figure 2.10: (a) cRBM architecture [6] where both current visible state and hidden state are conditioned on previous visible state to model the temporal dependencies. (b) FCRBM architecture [7] as an extension of cRBM where the interactions are gated by style features.

states similar to the hidden state in LSTMs. To model the stochastic nature of human motion, a small Gaussian noise was added to the biases of the hidden state at each step which encourages the model to explore different values in the hidden state space given the same inputs. This strategy also helped with the mean collapse problem defined as converging to the average of possible outputs due to the high uncertainty present in the human motion future poses, usually appearing as character floating. Although injecting noise helps the model avoid mean collapse, it is hard to tweak as it might cause discontinuous and noisy output. To increase the expressiveness power, one can stack multiple layers of cRBM, similar to Deep Belief Networks (DBN), where the sequence of hidden state vectors at the lower layer act as the fully observed data for the current layer.

In the initial version of cRBM, the control signal and movement styles were not explicitly provided to the model, and the motion could only be controlled by the provided initial frames. For example, if the initial frames were sampled from a walking sequence, the generated sequence resembled a walking movement during the future frames. To improve cRBMs in this direction, Taylor and Hinton [7] proposed Factored Conditional Restricted Boltzmann Machines (FCRBM) by introducing multiplicative three-way interactions where the dynamic state of a third unit (in this work style feature) modulates the interactions between other pairs of units (input, output,

and hidden units) (Figure 2.10b). The style features were computed by applying a linear layer to the one-hot encoding of different styles. Different styles can then be blended by interpolating style features. Factoring out the control parameters such as style or type of motion can also help avoid the convergence to the mean pose and floating problem. However, tweaking the trade-off between noisy motions and the mean collapse and floating character was still tricky.

2.2.2.3 Hidden Markov Models

Hidden Markov Models (HMM) [81, 82, 83] are classical statistical Markov models widely used in probabilistic sequential modelling such as speech synthesis, motion modelling, gesture recognition, etc. These models comprise two processes, hidden state and observable state, where the unobserved sequential hidden state is assumed to be Markov (defined by a state-to-state *transition probability*), and the conditional probability distribution of observable state given the history of hidden states only depends on the hidden state at the current time (defined by state-to-signal *emission probability*). The learning process is defined by driving the maximum likelihood estimate of the transition and emission probabilities parameters given the set of output sequences. In general, no tractable algorithm is known for an exact likelihood maximization. However, some variants of expectation-maximization algorithms such as Baum–Welch algorithm [84] or the Baldi–Chauvin algorithm [85] can be used to derive a local maximum likelihood efficiently. Integrating control or motion attributes to the HMMs can be done in different ways, such as conditioning the emission probability to the control factor [86, 87, 41], or training separate HMMs for each factor (attribute, style, etc) and deriving new HMMs by interpolating the trained individual HMMs [88]. HMMs require a hidden state size which is exponential in the number of components and therefore suffers from having a simple hidden state.

Brand and Hertzmann [41] proposed *Stylistic Hidden Markov Models* (SHMM) also known as *Style Machines* which defines a space of HMMs by incorporating a multidimensional style variable to vary the model parameters. Learning the parameterized model in Style Machines involves a simultaneous training of a generic HMM, modelling the generic behaviour present in the entire family of an action type, and a set of style-specific HMMs to capture the stylistic variations of the action type. The optimization is performed via Baum-Welch EM algorithm [84] to minimize the sum of generic model entropy, the cross-entropy between generic model and the statistics of

training data, and the cross-entropy between generic and style-specific models. The continuous style space was learned in an unsupervised manner. First, the initial space was constructed by the style-specific feature vectors of each style-specific HMM. The feature vectors were created by concatenating the state means, square root covariance, and the state dwell times (the average time that the model stays in one state before the transition to another. Then, the dimensionality of the space was reduced using PCA to extract the dimensions with the most variations. Finally, the new styles can be generated by interpolation or extrapolation, resulting in mixing or exaggerating styles, respectively.

2.2.2.4 Generative Adversarial Networks

Generative adversarial networks (GAN) [8] are considered one of the most important breakthroughs in the deep learning area. GAN was designed based on a game theory scenario in which a generator network competes with a discriminator network (Figure 2.11). While the generator network learns to generate plausible data, the discriminator network attempts to distinguish between the samples drawn from training data (real data distribution) and fake samples generated by the generator. The output of the discriminator network is the probability value $D(\boldsymbol{x};\boldsymbol{\theta}^{(d)})$, indicating the probability of x being drawn from real data distribution rather than a fake sample from the generator. $\boldsymbol{\theta}^{(d)}$ indicates the parameters of discriminator. The learning process in GANs can be formulated as a minimax two-player scenario where the generator attempts to maximize the probability of the discriminator making a mistake. In other words, generator and discriminator play the minimax game with the value function $v(\boldsymbol{\theta}^{(g)},\boldsymbol{\theta}^{(d)})$:

$$\min_G \max_D v(\boldsymbol{\theta}^{(g)},\boldsymbol{\theta}^{(d)}) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))], \tag{2.5}$$

where G is the generator and $\boldsymbol{\theta}^{(g)}$ is its set of parametes. The convergence in learning process is reached when the generated fake samples cannot be distinguished from real data by discriminator. In this case, discriminator outputs 1/2 for all samples. The above objective function may not provide sufficient gradients to the generator at the beginning of training since the discriminator rejects fake samples with high confidence. To address this, the original paper suggested training G to maximize $\log(D(G(\mathbf{z})))$ rather than to minimize $\log(1-D(G(\mathbf{z})))$.

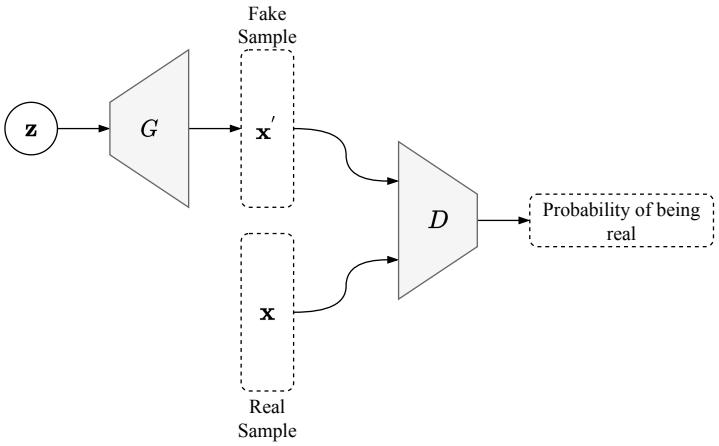


Figure 2.11: The graphical representation of generative adversarial networks architecture proposed by Goodfellow et al. [8].

Gui et al. [9] proposed *adversarial geometry-aware encoder-decoder* (AGED) model for forecasting human motion based on GANs architecture where the generator was constructed by a sequence-to-sequence model (called *predictor*). The sequence-to-sequence encoder-decoder generator was fed by a sequence of poses called *conditioning sequence* to produce a sequence of predicted future poses. Two global recurrent discriminators, *fidelity discriminator* and *continuity discriminator*, were introduced to validate the predicted sequence plausibility and its coherence with the conditioning sequence. In addition to the two adversarial losses, a frame-wise geodesic loss was proposed for more precise distance measurement to regress the predicted sequence of poses to the ground truth sequence. Figure 2.12 illustrates the architecture of the proposed adversarial geometry-aware encoder-decoder (AGED) model.

Barsoum et al. [89] proposed a sequence-to-sequence GAN model similar to [9] by combining the original GAN structure with a modification of the GAN scheme called *Wasserstein GAN* (WGAN) [90]. WGAN was integrated into the architecture by adding a *critic network* to the model. The WGAN loss was used to train the critic network and the generator network, while the GAN loss was used only to train the discriminator networks. In addition, instead of the continuity discriminator, a *consistency loss* or *pose gradient loss* was incorporated to penalize discontinuities between the consecutive posses.

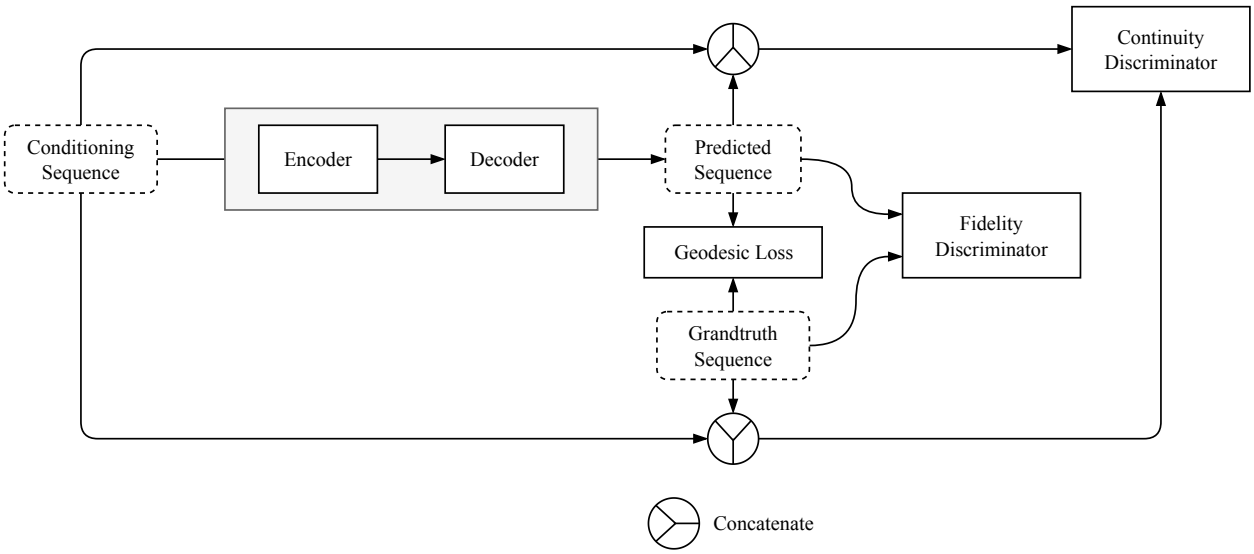


Figure 2.12: The graphical representation of architecture proposed by Gui et al. [9]. The final objective function comprises two adversarial losses (corresponding to two discriminators) and the geodesic loss.

2.2.2.5 Autoregressive Models

Autoregressive Models represent the joint distribution of data $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_D\}$ as the product of conditional distribution using the chain rules of probability,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \prod_{i=1}^{D} p_{\boldsymbol{\theta}} \left(\mathbf{x}_i \mid \mathbf{x}_{<i} \right) \tag{2.6}$$

where $\mathbf{x}_i$ is the i-th element of $\mathbf{x}$ in some pre-specified order, $\mathbf{x}_{<i}$ represent all elements before $\mathbf{x}_i$, and $\boldsymbol{\theta}$ is the set of parameters of the model. In case of time series data the order is defined across time conditioning each element on previous elements in time. The conditional distribution can be defined using different approaches. An early example is linear autoregressive models, which were later improved by the introduction of hidden-state models such as HMMs [91] and Kalman Filters [92]. Modern neural autoregressive models extended the ability of these models by modelling the highly non-linear dependencies of conditional distribution using neural networks. To set the computational complexity independent of the number of dimensions and also share information among different conditionals, the parameters can be shared among conditional distributions mainly using two approaches: 1) **Recurrent Neural Networks**, and 2) **Masking-based Models**.

Recurrent Neural Networks (RNNs): RNNs can also be seen in a probabilistic point of view when the output at each step is a distribution over possible outputs rather than a single output. RNNs rather than feeding directly all previous dimensions, $\mathbf{x}_{<i}$, into the conditional network $p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{x}_{<i})$, use a a fixed-length vector called *hidden state*, $\mathbf{h}_i$, which summarizes all previous dimensions.

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}) &= \prod_{i=1}^{D} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{x}_{<i}) \\ &= \prod_{i=1}^{D} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{h}_{i-1}) \,. \end{aligned} \tag{2.7}$$

Each time-step includes two main operations: updating the hidden internal state of the model which summarizes the past information, and a mapping from the hidden state to the next element in the sequence. Therefore, at each time-step we have

$$\mathbf{h}_i = f_h(\mathbf{h}_{i-1}, \mathbf{x}_i) \tag{2.8}$$

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}_{i+1}|\mathbf{x}_{\leq i}) &= p_{\boldsymbol{\theta}}(\mathbf{x}_{i+1}|\mathbf{h}_i) \\ &= f_o(\mathbf{h}_i), \end{aligned} \tag{2.9}$$

where f_h is the non-linear updating function parameterized by $\boldsymbol{\theta}$. f_o is the mapping from internal state to the output and usually characterized by another network.

Vanilla RNN was extended for addressing problems such as exploding/vanishing gradients and improving the long term memory by introducing *Long-Short Term Memory* (LSTM) cells [65] and its variations such as *Gated Recurrent Unit* (GRU)[63]. Many of the proposed methods [43, 67, 1, 2, 68] for modelling human motion is based on recurrent neural networks (RNN). However, since the only source of variability could occur when sampling the output, these models fail to reflect the stochasticity and variability observed in complex spatiotemporal data such as human motion. in Encoder-Recurrent-Decoder, Fragkiadaki et al. [1] considered both deterministic and probabilistic predictions. In the probabilistic case, the prediction of the next frame was modelled by a Gaussian Mixture Model (GMM) which achieved a small improvement over the deterministic prediction.

Masking-based Models: Masking-based models are the second branch of neural autoregressive models with a key property of parallelized computation of all conditionals. In a general form the

conditional distributions can be defined as

$$
\begin{aligned}
p_{\boldsymbol{\theta}}(\mathbf{x}) &= \prod_{i=1}^{D} p_{\boldsymbol{\theta}}\left(\mathbf{x}_i \mid \mathbf{x}_{<i}\right) \\
&= \prod_{i=1}^{D} p_{\boldsymbol{\theta}}\left(\mathbf{x}_i \mid \mathbf{x} \odot \mathbf{m}_i\right),
\end{aligned}
\tag{2.10}
$$

where $\mathbf{m}_i$ is a mask, masking current and future dimensions in the predefined order so the $\mathbf{x}_i$ will only depend on previous dimensions. In this formulation, the likelihood probability of each sample can be computed as a parallelized computation for all elements. However, during generation, the elements are computed recursively given the previous generated elements. An early type of such models was **MADE** proposed by Germain et al. [93], where the connections were established using a masked multilayer perceptron to enforce the autoregressive property in an autoencoder efficiently. MADE utilizes parameter sharing in computing the conditional distributions for a better generalization, faster convergence, and more efficient training. However, similar to fully connected networks it is not exploiting the spatial structure of data such as images.

In the context of temporal data, and in particular audio signals, Van Den Oord et al. proposed a masked-based architecture called **WaveNet** [10] that consists of a stack of causal convolution, which is a convolution operation designed to respect the ordering where the prediction at a certain time-step only depends on the data observed in the past. The convolution-based design allowed efficient parameter sharing with constant parameter count for variable-length distribution. One drawback of the convolution layer is the finite dynamic response (limited size of receptive fields), meaning that the output at each time-step only depends on a finite (and limited) number of previous time-steps, which prevents the network from modelling long-term dependencies. WaveNet, therefore, adopted a variation of convolutional operations called *dilated convolution* where the kernel is applied to an evenly-distributed subset of samples (Figure 2.13). This resulted in a much larger receptive field of inputs. Pavllo et al. [29] adopted a similar dilated causal convolutional network to improve their previous RNN-based architecture [43] in terms of training time and addressed some of the common problems in RNNs such as vanishing and exploding gradients. Van Den Oord et al. also proposed *PixelCNN* model [94] that utilizes a similar idea for generating conditional image generation by using a masked variant of convolutional neural networks, which exploits the spatial knowledge.

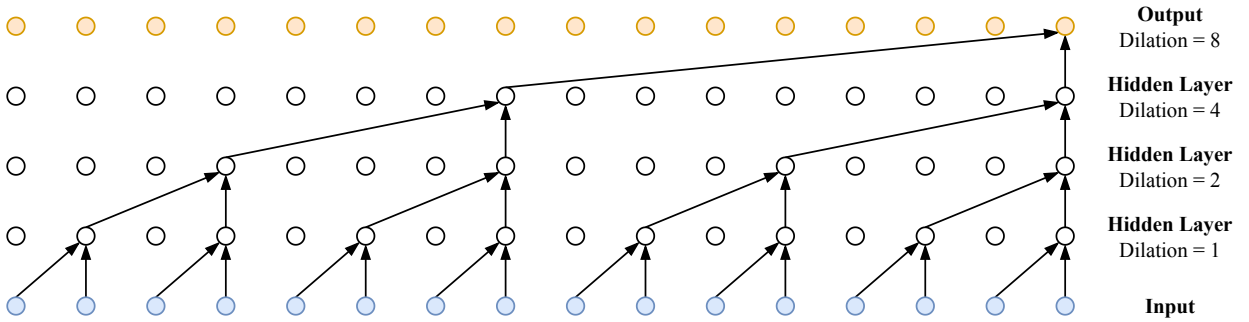


Figure 2.13: Visualization of a stack of dilated causal convolutional layers in Wavenet proposed by Van Den Oord [10].

Similar to WaveNet and MADE, the computations for evaluating each pixel during training can be performed in parallel. However, the synthesis is still sequential as each generated pixel is fed back to the network as input to compute the next pixel. PixelCNN was later improved by Salimans et al. [95] by adding a few modifications. They discretized Logistic distribution rather than a Softmax to speed up the training procedure. To capture the long-term dependencies, they used skip connections across the convolutional layers, which helped speed up the optimization.

In general, autoregressive models are state of the art in many modalities and datasets. Using masked-based models makes the training phase of autoregressive models efficient as all conditional probabilities can be computed using one forward pass. In addition, autoregressive models are expressive as the factorization is very general and can be generalized as the meaningful parameter sharing has a good inductive bias. However, the sampling is very slow as the forward pass in sampling is still performed recursively. Reed et al. [96] proposed a hierarchical multi-scale generation scheme in which certain pixel groups were conditionally independent to avoid requiring a full network evaluation per element. The same idea was exploited by Weissenborn et al. [97] for generating videos where the hierarchical multi-scale structure was used across both spatial and temporal directions. Caching hidden unit activations [10, 98] was another approach to alleviate the slow generation problem in autoregressive models where the hidden state computed in each step was cached into the memory to be used for later time-steps.

2.2.2.6 Variational Autoencoder

Before explaining Variational Autoencoders, lets have a brief description of **(Deep) Latent Variable Models**. Latent Variable models take an indirect approach for describing the data distribution by extending the fully-observed directed models into directed models with latent variables. *Latent variables*, typically denoted by $\mathbf{z}$, are the variables that are not observed in the data, but are part of the model underlying the process of how the data is generated. The intuition here is that under the hood there might be a relatively simple generation process that can be described with less variables and more abstract determinants. Hence, the model probability distribution is described as a marginalization of the joint probability of observed and latent variables as

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}) &= \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\ &= \int p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) d\mathbf{z}, \end{aligned} \tag{2.11}$$

which is called *model evidence*. Therefore, a very complex *model evidence* (or marginal likelihood) $p_{\boldsymbol{\theta}}(\mathbf{x})$ can be described by relatively simple parameterized *prior* $p_{\boldsymbol{\theta}}(\mathbf{z})$ and *likelihood* $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ distributions. In addition, we can make part of observation space independent conditioned on some latent variables which results in a faster sampling compared to autoregressive models by exploiting statistical patterns.

One example of such models is the mixture-of-Gaussians distribution, where the latent variable $\mathbf{z}$ is discrete and the likelihood $p(\mathbf{x} \mid \mathbf{z})$ is a Gaussian distribution with a mean and covariance which depends on the value of the latent variable.

Deep Latent Variable Models (DLVM) are a class of latent variable models whose distributions are parameterized by neural networks. To learn such models we can maximize the negative log-likelihood where $p_{\boldsymbol{\theta}}(\mathbf{x})$ for each sample is computed by Eq. 2.11. The exact objective is possible when the latent variable $\mathbf{z}$ takes a small number of values. However, we are usually interested in continuous latent variable which usually makes the marginal probability of data intractable since there is no close form solution for $p_{\boldsymbol{\theta}}(\mathbf{x})$ in Eq. 2.11. Since the joint distribution, $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z})$, is intractable, the intractability of $p_{\boldsymbol{\theta}}(\mathbf{x})$ is directly related to the intractability of posterior.

Kingma et al. [11] proposed using *variational approximation* where the true posterior is ap-

proximated by a simple parametric form $q(\mathbf{z})$ such as Gaussian distribution for each data sample. To have a good approximation, we need $q(\mathbf{z})$ to be as close as possible to the true posterior for each data sample. Therefore, for each $\mathbf{x}^i$:

$$\min_{q(\mathbf{z})} \mathrm{KL}\left(q(\mathbf{z}) \| p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}^{(i)})\right). \tag{2.12}$$

Kingma et al. [11] also proposed using *amortized inference* by utilizing the same parameterization (a neural network) $q_{\phi}(\mathbf{z} \mid \mathbf{x})$ which takes data samples as input and outputs the parameters (mean and the variance if Gaussian distribution is used) of the corresponding posterior distribution. In this way, there is no need to run the optimization from scratch and separately for each data sample. This results in a faster optimization and an implicit regularization since the relationship between observed and latent variables is through a network that has some smooth structure to it. The amortized formulation is as follows:

$$\min_{\phi} \sum_{i} \mathrm{KL}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p_{\boldsymbol{\theta}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right)\right). \tag{2.13}$$

Now by having the *inference model*, $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$, the intractable posterior inference and the learning procedure can be turned into a tractable problem. The KL divergence in Eq. 2.13 can be written as

$$\mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})\right] = \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})\right] + \log p_{\boldsymbol{\theta}}(\mathbf{x}). \tag{2.14}$$

By rearranging the above equation we have

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{z}) + \log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})\right]}_{\text{Variational Lower Bound}} + \underbrace{\mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})\right]}_{\geq 0}, \tag{2.15}$$

where the first term in RHS is called *Variational Lower Bound* (VLB) or *Evidence Lower BOund* (ELBO). The second term is the gap between the true and approximate posteriors which cannot be computed in a close form. Therefore, we can instead maximize the ELBO which pushes up the log-likelihood $\log p(\mathbf{x})$ as well.

Finally variational parameters $\boldsymbol{\phi}$ and the generative parameters $\boldsymbol{\theta}$ can be optimized jointly by

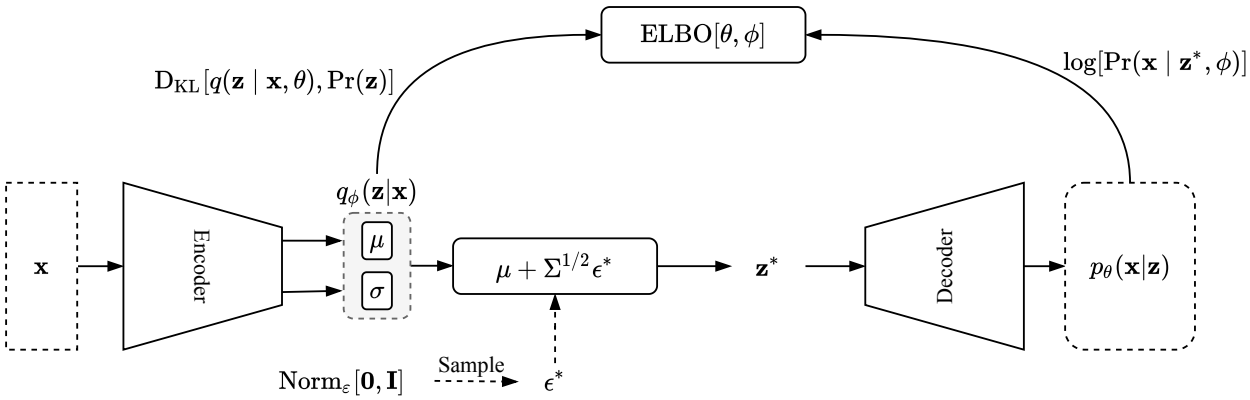


Figure 2.14: An overview of the original VAE model proposed by Kingma et al. [11].

maximizing the ELBO defined as follows:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbf{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})\right). \tag{2.16}$$

The first term is the expected log-likelihood or *reconstruction term* which is estimated by Monte Carlo estimator and reparameterization trick [11]. The second term, $\mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right)$, is the Kullback-Leibler divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z})$ which acts as a specific *regularization term* [99]. The combination of inference q_ϕ and generative p_θ models form a kind of autoencoder where the first term in Eq.2.16 is called the reconstruction term and the second term is called the regularization term. The diagram of the originally proposed architecture for VAEs is illustrated in Figure 2.14

Vanilla VAEs suffer when the space of outputs is multi-modal, resulting in blurry generated samples. Sohn et al. [100] proposed conditional VAE (CVAE) as an extension of VAE for learning structured output predictions simply by conditioning the generative process on an input variable. CVAE not only helps to address the problem of one-to-many mapping but also allows the model to control the input variable's generated sample class and/or characteristics. The ELBO for CVAE is formulated as follows:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}, \mathbf{a}) = \mathbf{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{a})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, \mathbf{a}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{a}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{a})\right), \tag{2.17}$$

where $\mathbf{a}$ is the input variable.

Since the introduction of the variational autoencoder, many different variations have been pro-

posed to improve it in different directions. Refer to [99] for more information and discussion of some of the variations of VAEs. In the following, we briefly describe the most common ways of modelling sequential data using Variational Autoencoder.

Recurrent Variational Autoencoder: Habibie et al. [101] used a variational autoencoder (VAE) to model the spatial relationships. However, they extended their model to operate in an autoregressive fashion by setting the cell state of the LSTM components equal to the corresponding latent variable at each time step during training. While this approach successfully couples the LSTM representation to the posterior distribution, collapsing the latent variable and internal state to one variable limits the representational power of the model's internal state. Additionally, the balance their architecture strikes between control signals and previous cell state during generation limits model performance for non-periodic complex movements. We attempt to mitigate these drawbacks in our proposed model by formulating the internal state and latent code in two separate channels and conditioning the latent code to the previous internal state in order to model the temporal dependencies during test time.

Motion VAE: Ling et al. [12] et al. proposed a model based on VAEs where the motion is controlled by setting the latent code as the output of a deep reinforcement learning module. They modelled the motion by a Markovian assumption meaning that each pose only depends on the previous pose, and the autoregressive model is memoryless. They also modelled the VAE decoder as a Mixture of Experts (MoE) network. Motion VAE has two main parts. An encoder that takes the previous x_{t-1} and current x_t states as input and outputs the parameters of the posterior distribution. Then the decoder takes the previous state and the sample latent variable to reconstruct the current state. The decoder was designed based on the Mode-Adaptive Neural Networks (MANN) [5] architecture with a mixture-of-experts neural network. Figure 2.15 shows an overview of MotionVAE architecture.

RNN-VAE: One of the common ways of modelling temporal and spatiotemporal data using VAEs is to construct the encoder and decoder networks of VAE by recurrent neural networks. The latent code can be computed as a function of all outputs at all steps of RNN or as a function of the last hidden state similar to sequence-to-sequence models. This provides a probabilistic framework for time-series prediction or machine translation.

Modelling temporal dependencies in encoder and decoder is not limited to RNNs or their vari-

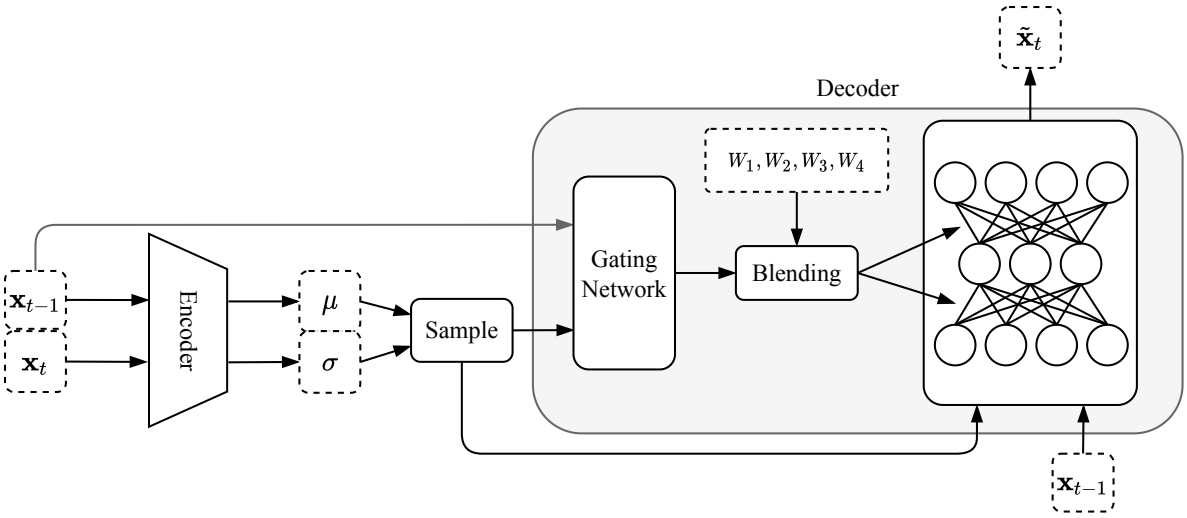


Figure 2.15: An overview of MotionVAE proposed by Ling et al. [12]

ations. Chen et al. [102] proposed *Latent ODE* approach where a *Neural Ordinary Differential Equations* (Neural ODE) model was used to replace the decoder as a continuous-time generative model to modelling time series. The latent ODE model is suitable for applications where the time-series data are irregularly sampled and provides a way of interpolation and extrapolation at arbitrary time points. Rubanova et al. [103] extended Latent ODE to a fully ODE-based sequence-to-sequence model where the encoder is constructed by an ODE-RNN network. ODE-RNN network is similar to normal RNNs but with an ODE updating the hidden state of the RNN.

Similar to normal VAEs, RNN-VAEs can also incorporate control signals to control the generated sequences. Greenwood et al. [13] proposed a model for speech-driven head movements using conditional RNN-VAE that was originally proposed by Walker et al. [104] for predicting the dense trajectory of pixels in a scene (what will move in the scene). In the conditional RNN-VAE, the input to the encoder is the sequence of input features concatenated to the control signals and the input to the decoder is the concatenation of the control signal and sampled latent code. Fig. 2.16 illustrates the conditional RNN-VAE architecture proposed by [13] and [104].

Variational Recurrent Neural Network (VRNN): Variational Recurrent Neural Network (VRNN) was first proposed by [14] where the latent random variable was included in the hidden state of a recurrent neural network by combining the elements of the variational autoencoder. The use of high-level latent random variables was proposed to model the variability observed in stochastic highly structured data such as speech data. The authors characterized "highly structured

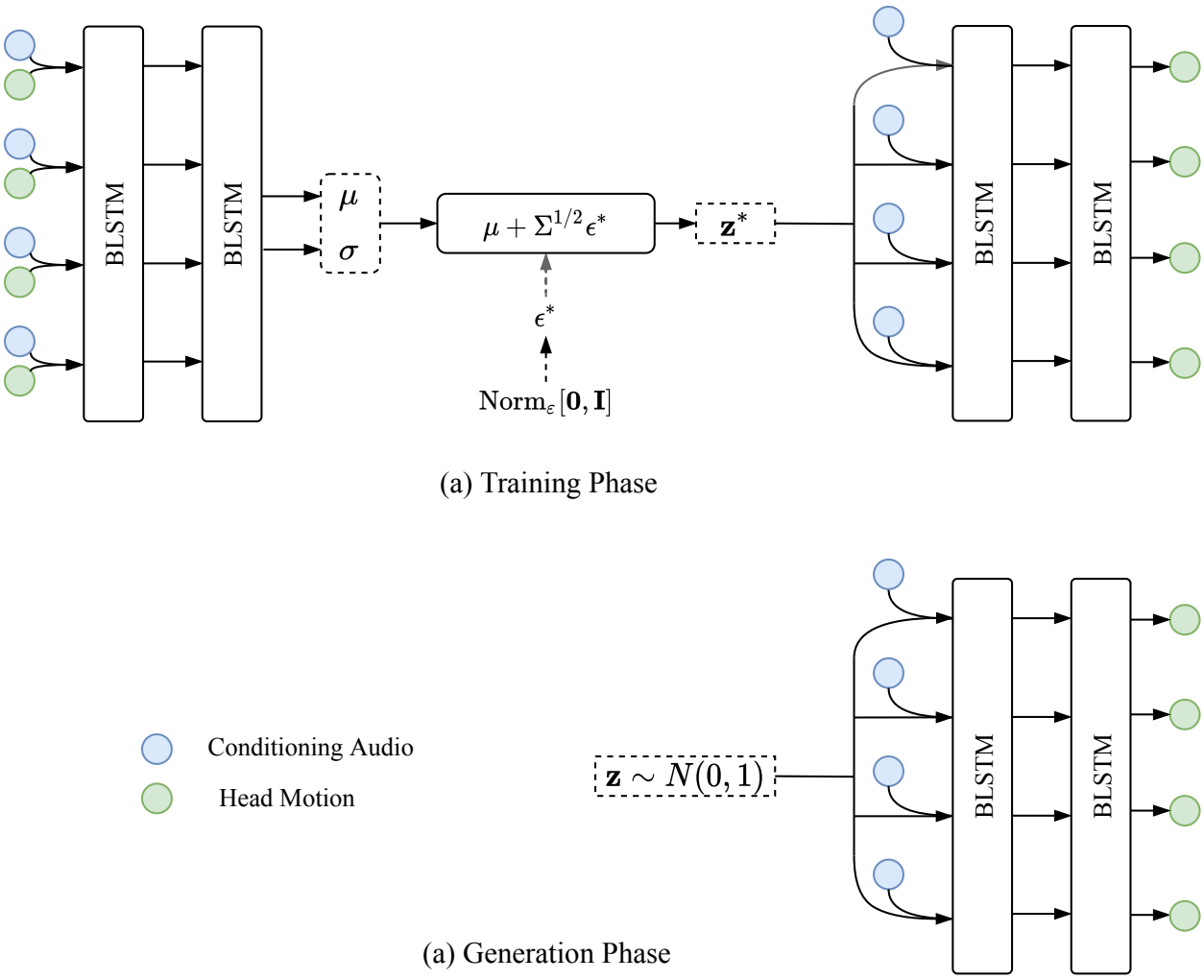


Figure 2.16: An overview of conditional RNN-VAE model proposed for synthesizing head motion given speech data by Greenwood et al. [13]. During the generation phase, sampling from prior adds natural variations to the generated head movements while being controlled by the conditioning audio.

data" by two properties. First, there is a high signal-to-noise ratio in the data. This means that the source of the major variability observed in data is the signal itself and not noise. Second, there exists a highly non-linear complex relationship between the factors of variation and the observed data. Such assumptions lend themselves well to the deep latent variable models where the latent random variables can model the variations observed in data, and a neural network that models the complex dependencies between the underlying factors of such variations and observed data.

The structure of architecture for each recurrent step in VRNN is illustrated in Figure 2.17. Each step contains VAE conditioned on the previous hidden state ($\mathbf{h}_{t-1}$) of the integrated RNN to model

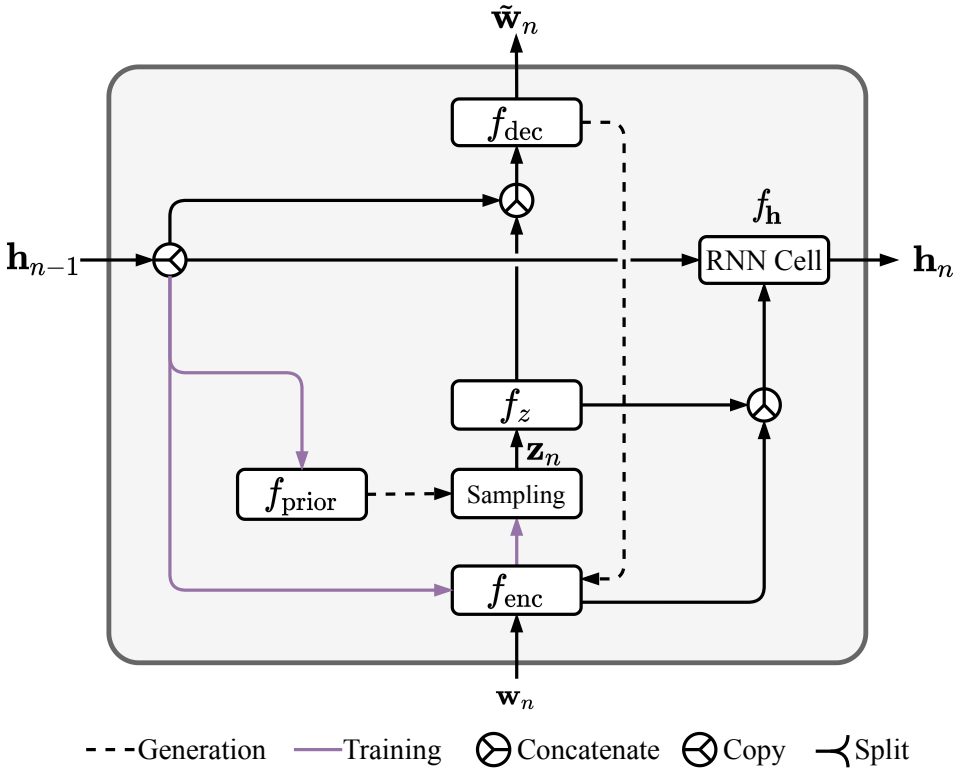


Figure 2.17: An overview of conditional VRNN model at one time-step proposed by Chung et al [14]. f_{enc}, f_{dec}, f_z, f_{prior} are arbitrary functions such as neural networks.

the temporal dependencies in the sequential data. Conditioning on the previous hidden state is constructed by conditioning prior, posterior, and likelihood terms in the VAE to the previous hidden state. At each time step, the hidden state of the RNN is updated by observed data at the current step, the previous hidden state, and the sampled latent variable. During the generation phase, the input to the decoder is not only the sampled latent variable but also the previous hidden state. The sampling process acts as the stochastic component of the model which adds the variability during generation. The objective function in VRNN is defined as the sum of step-wise ELBO across all steps:

$$\mathbb{E}_{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})}\left[\sum_{t=1}^{T}\left(-\mathrm{KL}\left(q\left(\mathbf{z}_t \mid \mathbf{x}_{\leq t}, \mathbf{z}_{<t}\right) \| p\left(\mathbf{z}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{<t}\right)\right) + \log p\left(\mathbf{x}_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}\right)\right)\right], \tag{2.18}$$

where conditioning to the previous steps in all terms is provided by conditioning to the previous hidden state, as it summarizes observed data and latent variables of the all previous steps. The parameters of posterior and prior distributions are provided by f_{enc} and f_{prior}, respectively. f_{dec} is

the decoder component of the VAE and f_z is applied on latent variable as a feature extractor.

2.2.2.7 Normalizing Flows

Normalizing flows [105, 106] are a class of generative models which are characterized based on the **change-of-variables formula** for probability density functions. The goal here is to transform a simple distribution into a complex one by applying a sequence of invertible transformation functions. Suppose $\mathbf{g}_{\boldsymbol{\theta}}$ is a bijective mapping $\mathbf{g}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^D$, parameterized by $\boldsymbol{\theta}$. $\mathbf{z} \in \mathbb{R}^D$ is a random variable, and $\mathbf{x} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{z}) \in \mathbb{R}^D$ is the random variable constructed by applying $\mathbf{g}_{\boldsymbol{\theta}}$ on $\mathbf{z}$. Given the probability density of $\mathbf{z}$, one can compute the probability density of $\mathbf{x}$ using change of variable formula:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}) &= p(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right|^{-1} \\ &= p_Z(\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\mathbf{x})) \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right) \right|^{-1} \\ &= p_Z(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})) \left| \det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right|, \end{aligned} \tag{2.19}$$

where $p_Z(\mathbf{z})$ is a simple distribution such as Gaussian, $\mathbf{f}_{\boldsymbol{\theta}}$ is the inverse of $\mathbf{g}_{\boldsymbol{\theta}}$, and $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ and $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ are the Jacobian of functions $\mathbf{g}_{\boldsymbol{\theta}}$ and $\mathbf{f}_{\boldsymbol{\theta}}$, respectively. For the above equation to be true, $\mathbf{g}_{\boldsymbol{\theta}}$ (and clearly $\mathbf{f}_{\boldsymbol{\theta}}$) has to be **invertible**. Such invertible transformations are also called *bijections*. This model can be optimized by maximizing the likelihood objective over all training data samples, which is of the form:

$$\max_{\boldsymbol{\theta}} \sum_i \log p_{\boldsymbol{\theta}} \left(\mathbf{x}^{(i)} \right) = \max_{\boldsymbol{\theta}} \sum_i \log p_Z \left(\mathbf{f}_{\boldsymbol{\theta}} \left(\mathbf{x}^{(i)} \right) \right) + \log \left| \det \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \mathbf{x}} \left(\mathbf{x}^{(i)} \right) \right|. \tag{2.20}$$

To optimize the above objective function, other key requirements for such transformations is to have an **easy-to-calculate Jacobian** and being **differentiable**. To generate new samples, we can use the transformation in the other direction where the sampling can be performed in two main steps: 1) sampling from latent space $\mathbf{z} \sim p_Z(\mathbf{z})$), 2) transforming from latent space to the data space ($\mathbf{x} = \mathbf{g}_\theta(\mathbf{z})$).

Considering the fact that the composition of bijections is itself a bijection, one can increase the complexity and expressivity of the transformation by composing multiple bijections:

$$\begin{aligned}\mathbf{x} &= \mathbf{g}_K \circ \mathbf{g}_{K-1} \circ \cdots \circ \mathbf{g}_1 (\mathbf{z}) \\ &= \mathbf{g}(\mathbf{z})\end{aligned} \tag{2.21}$$

where $\mathbf{g}_i$ is a simple bijection and $\mathbf{g}$ is itself a bijection. The inverse of $\mathbf{g}$ is

$$\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ \cdots \circ \mathbf{f}_K, \tag{2.22}$$

with a determinant of Jacobian as the product of the determinant of Jacobian of all bijections

$$\left| \det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right| = \prod_{i=1}^{K} \left| \det \frac{\partial \mathbf{f}_i(\mathbf{x}_i)}{\partial \mathbf{x}} \right|, \tag{2.23}$$

where $\mathbf{x}_i$ is the i-th intermediate output of flow composition. In general, there are many different types of flows. In the following, we briefly describe, *coupling flows*, as one of the main classes of normalizing flows and refer the reader to [107] for a detailed review of all classes of normalizing flows. We then describe some of the approaches proposed for modelling spatiotemporal data using normalizing flows.

Coupling Flows: Coupling Flows were introduced by Dinh et al [108, 109] as an efficient and expressive bijection for flow-based models. Given a D dimensional input $\mathbf{x} \in \mathbb{R}^D$, the output $\mathbf{z}$ of a coupling flow is computed as follows:

$$\begin{aligned}\mathbf{z}^A &= \mathbf{x}^A \\ \mathbf{z}^B &= \mathbf{h}\left(\mathbf{x}^B; \Theta(\mathbf{x}^A)\right)\end{aligned} \tag{2.24}$$

where $\mathbf{h}$ is a bijection (called *coupling function*) with parameters $\boldsymbol{\theta}$ defined by an arbitrary complex function Θ (called *conditioner*), and $(\mathbf{x}^A, \mathbf{x}^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$ are the result of a disjoint partitioning of the input. Each coupling layer keeps one subspace identical and applies a bijection function only on the other subspace. The inverse of above transformation is computed as follows:

$$\begin{aligned}\mathbf{x}^A &= \mathbf{z}^A \\ \mathbf{x}^B &= \mathbf{h}^{-1}\left(\mathbf{z}^B; \Theta(\mathbf{x}^A)\right).\end{aligned} \tag{2.25}$$

The coupling functions in coupling flows are typically defined as an element-wise operations resulting in a triangular transformation and simple determinant of Jacobian computation. We refer the reader to [107] for a detailed review of *scalar coupling functions*. *Affine Coupling* transformations proposed by Dinh et al [108, 109] are defined as:

$$\begin{aligned} \mathbf{z}^A &= \mathbf{x}^A \\ \mathbf{z}^B &= \mathbf{x}^B \odot \exp(\Theta_1(\mathbf{x}^A)) + \Theta_2(\mathbf{x}^A), \end{aligned} \tag{2.26}$$

with the inverse transformation as follows:

$$\begin{aligned} \mathbf{x}^A &= \mathbf{z}^A \\ \mathbf{x}^B &= (\mathbf{z}^B - \Theta_2(\mathbf{x}^A)) \odot \exp(-\Theta_1(\mathbf{x}^A)), \end{aligned} \tag{2.27}$$

and Jacobian of:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial \mathbf{z}^B}{\partial \mathbf{x}^A} & \operatorname{diag}\left(\exp(\Theta_1(\mathbf{x}^A))\right) \end{bmatrix}, \tag{2.28}$$

where $\Theta_1 : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\Theta_2 : \mathbb{R}^d \to \mathbb{R}^{D-d}$ are *scale* and *translation* functions defined as arbitrary complex functions such as neural networks, and $\odot$ is the element-wise or Hadamard product. Figure 2.18 illustrates the diagram of an affing coupling function. These coupling flows are usually preceded by *normalization* and *permutation* layers. The role of the normalization layer is to address some of the problems occurring during the training process. Two examples of normalization layers are *batch normalization* and *ActNorm layer* proposed in [109] and [110], respectively. The permutation layer increases the expressiveness of the model by reordering dimensions and letting the model exploit the correlation between different dimensions. For instance, [109] proposed checkerboard and channel-wise partitioning acting as a fixed permutation and [110] proposed a learnable invertible 1×1 convolution. A block of normalization-permutation-coupling is called a *Flow Step*. A more complex flow transformation can be constructed by stacking several Flow Steps (Figure 2.19). In the following, we review some of the approaches for modelling spatiotemporal data using coupling flows.

Spatiotemporal Conditioner in Coupling Flows: One of the most common approaches to model spatiotemporal dependencies using normalizing flows is to apply coupling flow on the data

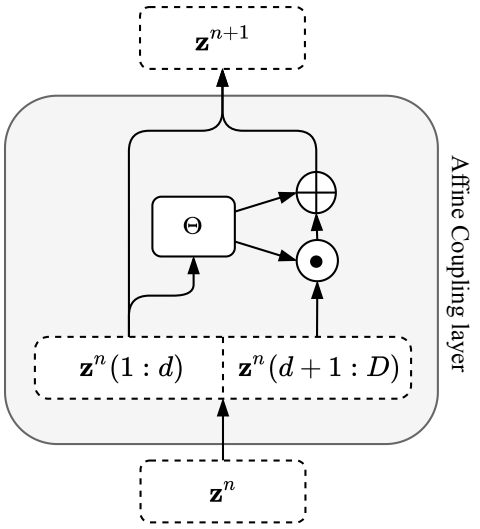


Figure 2.18: Block diagram of Affine Coupling layer

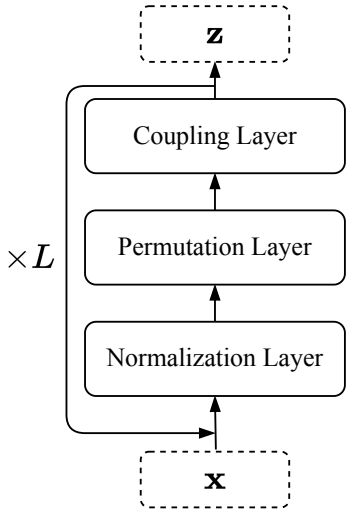


Figure 2.19: Stacking several Flow Steps resulting in higher expressivity

at each time-step and conditioning the conditioner on not only the other partition at the current time step but also previous time steps as well. Let $\mathbf{x} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}\}$ be a discrete spatiotemporal data sequence where $\mathbf{x}_t \in \mathbb{R}^D$ represents observed data at time-step t. Also, let $\mathbf{z} = \{\mathbf{z}_{0,l}, \mathbf{z}_{1,l}, \ldots, \mathbf{z}_{T-1,l}\}$ be the sequence of corresponding latent variables at l-th flow step, where $\mathbf{z}_t, l \in \mathbb{R}^D$. The coupling flow at time-step t can be expressed as:

$$\begin{aligned} \mathbf{z}^A_{t,l} &= \mathbf{z}^A_{t,l-1} \\ \mathbf{z}^B_{t,l} &= \mathbf{h}\left(\mathbf{z}^B_{t,l-1}; \Theta(\mathbf{z}^A_{t,l-1}, \mathbf{x}^A_{\leq t})\right), \end{aligned} \tag{2.29}$$

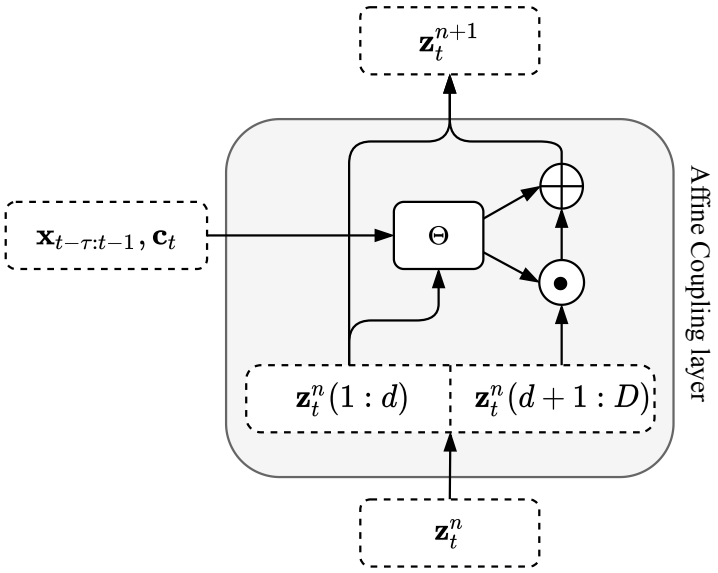


Figure 2.20: The coupling flow structure used in MoGlow, WaveGlow, and FlowWaveNet. The conditioner was defined by and LSTM for MoGlow and by a WaveNet architecture for WaveGlow and FlowWaveNet.

where the conditioner Θ can be defined as an RNN or an autoregressive network. Figure 2.20 demonstrates the base structure of a flow step for this approach.

Henter et al. [27] proposed a probabilistic flow-based model for modelling human motion with an architecture inspired by WaveGlow [111] and FloWaveNet [112] which were proposed for audio waveforms. They modelled the spatial dependencies by a Glow-like [110] structure on the space of poses and the temporal dependencies by conditioning the conditioner function of coupling layers on the past information at each time step. The conditioner function is modelled as an LSTM cell which takes not only half of the dimensions of current activation but also τ previous steps of poses and control signals as input at each time step. In this way, the previous information is provided by the hidden state and the τ previous poses. During generation, the latent code is sampled from the based distribution at each time-step and independent of other time-steps. Therefore, this sampling acts as injecting the stochasticity at each time-step resulting in producing different variations of output given the same control signal and initialization. The coupling layer of l-th flow step in MoGlow can be expressed as:

$$\begin{aligned} \mathbf{z}_{t,l}^A &= \mathbf{z}_{t,l-1}^A \\ \mathbf{z}_{t,l}^B &= \mathbf{x}_{t,l-1}^B \odot \exp(\Theta_1) + \Theta_2, \end{aligned} \tag{2.30}$$

where Θ_1 and Θ_2 are computed as follows:

$$\begin{aligned} \mathbf{h}_{t,l} &= \mathbf{LSTMCell}_l((\mathbf{z}^A_{t,l-1}, \mathbf{x}_{t-\tau:t-1}, \mathbf{c}_{t-\tau:t}), \mathbf{h}_{t-1,l}) \\ (\Theta_1, \Theta_2) &= \mathbf{Linear}_l(\mathbf{h}_{t,l}) \end{aligned} \tag{2.31}$$

where $\mathbf{h}_{t,l}$ is the hidden state of the LSTM cell at layer l and time-step t, and $\mathbf{c} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_T\}$ is the sequence of control signals. The Jacobian also is expressed as Eq. 2.28. The temporal dependencies do not necessarily need to be causal. FlowWaveNet [112] and WaveGlow [111] used a similar structure for speech synthesis and applied a non-causal WaveNet [10] as a conditioner at each layer where future time-steps are also fed to the conditioner.

Alexanderson et al. [33] proposed a flow-based generative model for speech-driven gesticulation using a similar architecture from their previous work MoGlow [27]. The control signal was a sub-sequence of acoustic features synchronized with the pose sequence. They also augmented the control signal with an optional style input to generate motions matched with the desired output style where they used hands height, hands speed, symmetry, and gesticulation radius as style control which is highly correlated with the speech-driven gestures.

Autoregressive Factorization of Latent Variable: In the proposed framework in MoGlow, the base distribution at each time-step is a fixed standard normal distribution. Therefore, during synthesis, sampling from base distribution at each time-step solely acts as a noise injection procedure resulting in a stochastic synthesis while the sequential modelling is performed in the coupling flows. Another way of modelling temporal dependencies in normalizing flows is to model the base distributions as an autoregressive factorization as follows:

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \prod_{t=1}^{T} p_{\boldsymbol{\theta}}\left(\mathbf{z}_t \mid \mathbf{z}_{<t}\right). \tag{2.32}$$

For instance we can let each $p_{\boldsymbol{\theta}}\left(\mathbf{z}_t \mid \mathbf{z}_{<t}\right)$ to be a conditional factorized Gaussian distribution with the parameters computed as follows:

$$p_{\boldsymbol{\theta}}\left(\mathbf{z}_t \mid \mathbf{z}_{<t}\right) = \mathcal{N}\left(\mathbf{z}_t; \boldsymbol{\mu}, \boldsymbol{\sigma}\right), \tag{2.33}$$

$$\text{where } (\boldsymbol{\mu}_t, \log \boldsymbol{\sigma}_t) = f_{\boldsymbol{\theta}} (\mathbf{z}_{<t}), \tag{2.34}$$

where $\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t$ are the mean and standard deviation of the base distribution at time-step t, and $f_{\boldsymbol{\theta}}$ is an arbitrary function parameterized by θ. In this formulation, both noise injection and sequential modelling are happening at the base latent variable level.

Kumar et al. [113] proposed this model for video synthesis and extended it by using a multi-scale architecture proposed by [109]. In a multi-scale architecture, half of the dimensions are factored out at regular intervals. Then the final latent variable is composed by stacking all factored out dimensions and the final remaining dimensions. In the sequential framework proposed by [113] (also called *VideoFlow*), the multi-scale architecture described above was used to infer the latent variable at each time-step $\mathbf{z}_t = \left\{ \mathbf{z}_t^{(l)} \right\}_{l=1}^{L}$, where $\mathbf{z}_t^{(l)}$ is the latent variable component at scale level l. Then, the conditional prior $p_{\boldsymbol{\theta}} (\mathbf{z}_t \mid \mathbf{z}_{<t})$ was specified as follows:

$$p_{\boldsymbol{\theta}} (\mathbf{z}_t \mid \mathbf{z}_{<t}) = \prod_{l=1}^{L} p_{\boldsymbol{\theta}} \left(\mathbf{z}_t^{(l)} \mid \mathbf{z}_{<t}^{(l)}, \mathbf{z}_t^{(>l)} \right), \tag{2.35}$$

where each $p_{\boldsymbol{\theta}} \left(\mathbf{z}_t^{(l)} \mid \mathbf{z}_{<t}^{(l)}, \mathbf{z}_t^{(>l)} \right)$ can be defined as a conditional factorized Gaussian distribution similar to Eq.2.33 and Eq.2.34.

2.2.3 Evaluation Methods

Most methods proposed for both short-term prediction (which are usually proposed in the computer vision community) and long-term generation use quantitative evaluations to assess the precision of the output. This is usually done by computing the Euclidean L_1 of L_2 norm of the distance between generated future poses and the grand truth data on joint angles or joint positions. The accuracy of such evaluation may be improved by adding joint velocity and acceleration which better reflect other criteria such as smoothness and continuity of the motion [4, 5, 44, 32, 114].

The metrics based on Euclidean norms do not represent the geodesic distance of two rotations which may confuse the training process, especially for large angular distances and at the beginning of the optimization process. One potential improvement in this direction is to define geodesic distance functions that respect the intrinsic structure of 3D rotations [9, 30].

Although the distance-based quantitative evaluations might be suitable for short-term prediction, such measures fail to assess long-term motion generation as they do not precisely measure the **quality** of the generated samples. In addition, such deterministic quantitative evaluation does not take into account the stochastic nature of human motion and its different variations given the same input, which we call **diversity**. While such quantitative metrics may give a good measure of how reliably the model can generate movements that are consistent with the control signals (for example, how well a character follows a trajectory), they fail to measure the believability and how convincing and natural are the generated motions.

A number of works used qualitative evaluations where human participants assess the perceptual qualities of motions generated by the proposed model and other baselines along with the real data to compare. The assessment can be based on motion quality, motion realism, motion variations, and consistency with the conditioning attributes and control signals. In such perceptual qualitative studies, a subset of samples from models and baselines are presented to the participants to rate, rank, or compare based on the instructed criteria.

In a recent work [30] we proposed using Inception Score (IS) [115] and Fréchet Inception Distance Score (FID) [116] which were originally proposed in the domain of images as an attempt to remove the subjective human evaluation of synthesized images. Both evaluation metrics have been shown to correlate well with human evaluation of generated images. Ideally, in a motion synthesis scenario, we evaluate models based on two main criteria: quality and diversity. First, we expect the generated samples to be realistic and coherent with the attributes which are set as control parameters (quality). Second, we expect the model to generate motions with a high diversity and natural stochasticity while still following the manifold of realistic motions (diversity). IS and FID attempt to codify both criteria. IS is formulated based on these two criteria defined as follows:

$$\text{IS} = \exp\left(\mathbf{E}_{\tilde{\boldsymbol{X}}\sim p_g} D_{KL}(p(\mathbf{a}|\tilde{\mathbf{X}})\|p(\mathbf{a}))\right) \tag{2.36}$$

where $\tilde{\mathbf{X}}$ is a synthetic sample generated by a generative model, $p(\mathbf{a}|\tilde{\mathbf{X}})$ is the conditional attribute distribution of a classifier which is pre-trained on separate training data, and $p(\mathbf{a}) = \int_{\tilde{\mathbf{X}}} p(\mathbf{a}|\tilde{\mathbf{X}}) p_{\text{model}}(\tilde{\mathbf{X}})$ is the marginal attribute distribution. Here, attributes are high-level labels such as action type or motion style class. Although IS is able to evaluate the model in terms of

both quality and diversity, samples should be paired with holistic attributes which is not the case in many scenarios.

FID captures the similarity between the generated and the real motion sample distribution. It evaluates the model by comparing the statistics of a set of generated samples to a set of real motion sequences from the dataset. Similar to IS, we use a classifier trained on a separate dataset. Then, the activations of an intermediate layer are summarized as a multivariate Gaussian distribution for synthetic and real data. The distance between the two distributions is then computed with Fréchet Distance as follows:

$$\text{FID} = \|\mu_g - \mu_d\|_2^2 + \text{Tr}\left(\Sigma_g + \Sigma_d - 2\left(\Sigma_g \Sigma_d\right)^{\frac{1}{2}}\right), \tag{2.37}$$

where $\mathcal{N}(\mu_g, \Sigma_g)$ and $\mathcal{N}(\mu_d, \Sigma_d)$ are the distributions of the activations in the last feature extraction layer for synthetic and real data, respectively. It has been shown that FID mimics human perception and judgement in discriminating the images, however, it has a strong assumotion of Gaussian distribution for the features [117].

Chapter 3

MoVi: A Large Multi-Purpose Human Motion and Video Dataset

The work in this chapter has been published previously as the following:

Saeed Ghorbani, Kimia Mahdaviani, Anne Thaler, Konrad Kording, Douglas James Cook, Gunnar Blohm, and Nikolaus F. Troje, "MoVi: A Large Multi-Purpose Human Motion and Video Dataset", in *PlosOne*, 2021

and has been updated and extended here for the purposes of this document.

3.1 Preface

Large high-quality datasets of human body shape and kinematics lay the foundation for modelling and simulation approaches in computer vision, computer graphics, and biomechanics. Creating datasets that combine naturalistic recordings with high-accuracy data about ground truth body shape and pose is challenging because different motion recording systems are either optimized for one or the other. We addressed this issue in our dataset by using different hardware systems to record partially overlapping information and synchronized data that lends itself to transfer learning. This multimodal dataset contains 9 hours of optical motion capture data, 17 hours of video data from 4 different points of view recorded by stationery and hand-held cameras, and 6.6 hours of inertial measurement units data recorded from 60 female and 30 male actors performing a collection of 21 everyday actions and sports movements. The processed motion capture data is also available as realistic 3D human meshes. In addition to human motion modelling, this dataset can be used for research on human pose estimation, action recognition, gait analysis, and body shape reconstruction.

3.2 Introduction

Capturing, modelling, and simulating human body shape and kinematics has been an area of intense study in the fields of biomechanics, computer vision, and computer graphics, with applications including human-machine interactions [118], assistive healthcare [119], clinical diagnostics [120], and realistic computer animation pipelines [121, 30, 20]. In order to obtain body pose and kinematics at a resolution that is fine enough to make inferences about identity, action, and particularly stylistic features, we need large, high-quality datasets that can be used in both generative and discriminative contexts. An unsolved challenge is to create datasets that combine video recordings of humans in motion in unconstrained scenarios with information on the ground truth about the dynamic pose and shape of the recorded individuals.

Research in computer vision has focused on understanding humans and their behaviour from images or videos. Obtaining reliable, high-accuracy data about the "true" pose and shape and its changes over time, however, requires sensors that might interfere with the ecological validity of the

image or video. For instance, optical motion capture has the potential to provide 3D pose and body shape [122], but conflicts with wearing normal clothing, leaves visible markers in the video, and can only be used in a laboratory environment. Other sensors, such as inertial measurement units (IMU), can be hidden under clothing and are feasible to capture humans in natural settings, but do not provide absolute location information and suffer from drift. One approach to eliminating this drift in IMU data is to detect the 2D joints of the body in a simultaneously recorded video [123]. Thus, limitations of one hardware system can partially be overcome by combining it with recordings of another.

Currently, no available single hardware system is able to capture people in a natural setting and simultaneously provide high precision ground truth data of body shape and pose. All publicly available datasets suffer from this limitation to some degree [124, 125, 126, 127, 128]. Some are also limited in that they either contain data of only a small number of different actors, use single hardware systems for motion recording, or provide unsynchronized data across different hardware systems. We address these limitations in our dataset by providing subsets of data with partially overlapping information that lend themselves to transfer learning. Our dataset contains five different subsets of synchronized and calibrated video, optical motion capture, and IMU data. Each subset features the same 90 female and male actors performing the same set of 20 predefined everyday actions and sports movements, plus one self-chosen movement.

An important advantage of our dataset is that the full-body motion capture recordings are also available as realistic 3D human meshes represented by a rigged body model as part of the AMASS database [122]. Because we recorded the same actors with varying combinations of sensors, these animated meshes can also be used as ground truth body shapes for the recording subsets with sparse markers and natural clothing. In addition to the MoSh++ formulation used in AMASS, we calculated the skeletal pose using the biomechanics formulation provided by the Visual3D software [18]. The synchronized and calibrated motion capture system and stationary video cameras allow computing and augmenting accurate frame-by-frame overlay of 3D skeletal pose and body surface in camera and motion capture coordinates. For our natural clothing captures, we recorded the motions using IMU sensors and video cameras, with and without additional sparse motion capture markerset. The sparse optical markerset could be combined with the IMU data to accurately extract end-effector locations and infer body pose.

This multi-modal dataset is designed for a variety of challenges including gait analysis, human pose estimation and tracking, action recognition, motion modelling, and body shape reconstruction from monocular video data and different points of view. To our knowledge, this is one of the largest datasets in terms of the recorded number of actors and performed actions, and the first dataset with the synchronized pose, pose-dependent and pose-independent body shape, and video recordings. The fact that we recorded the same actions from the same actors with varying combination of sensors makes our dataset unique.

3.3 Methods

3.3.1 Subjects

90 people (60 women, 30 men) with no reported neurological or musculoskeletal conditions that affected their ability to perform common sports movements were recruited from the local Kingston community. Participant characteristics are provided in Table 3.1. The experimental procedure was approved by the General Research Ethics Board of Queen's University, Kingston, and was performed in accordance with the Declaration of Helsinki. All participants provided written informed consent that their data (including their video footage) can be used by other researchers.

3.3.2 Acquisition Setup

An optical motion capture system, stationary and hand-held video cameras, and inertial measurement unit (IMU) sensors were used to record the dataset. Figure 3.1 shows the top-view floor plan of the capture room with the motion capture and video cameras arranged to cover a space of approximately 3 by 5 meters to allow subjects to perform their movements without spatial restrictions. In the following sections, the details of the hardware and software systems along with their calibration and synchronization process details are provided.

3.3.2.1 Hardware and Software Systems

Optical Motion Capture System. 15 Qualisys Oqus 300 and 310 cameras (Qualisys AB, Sweden, https://www.qualisys.com/) were used. The cameras were set to normal mode (full field of view) with a resolution of 1.3 MP and captured the 3D location of passive reflective markers of 0.7

Table 3.1: Participant characteristics of the 60 women and 30 men.

Women						Men					
ID	Age	Height [cm]	Weight [kg]	BMI [kg/m^2]	Handedness	ID	Age	Height [cm]	Weight [kg]	BMI [kg/m^2]	Handedness
2	33	152	54	23.37	right	1	25	184	92	27.17	right
6	26	155	59	24.56	right	3	26	167	59	21.16	right
7	22	175	73	23.84	right	4	26	178	80	25.25	right
8	22	160	52	20.31	right	5	23	180	73	22.53	right
9	23	157	48	19.47	right	11	27	178	90	28.41	right
10	24	175	63	20.57	right	13	26	178	77	24.30	right
12	26	162	68	25.91	right	15	21	181	72	21.98	right
14	21	157	61	24.75	right	18	25	170	65	22.49	right
16	26	163	68	25.59	right	19	18	167	60	21.51	left
17	26	167	65	23.31	right	20	29	173	60	20.05	right
21	21	160	55	21.48	right	22	28	170	66	22.84	right
24	20	160	55	21.48	right	23	25	173	73	24.39	right
25	21	166	55	19.96	right	26	24	178	63	19.88	right
30	19	178	68	21.46	right	27	23	163	64	24.09	right
32	20	168	57	20.20	right	28	25	183	80	23.89	right
34	21	155	41	17.07	left	29	24	177	61	19.47	right
38	32	157	53	21.50	right	31	28	175	64	20.90	right
39	21	175	77	25.14	right	33	21	175	60	19.59	right
40	21	175	56	18.29	right	35	29	176	72	23.24	right
44	20	162	75	28.58	right	36	29	174	74	24.44	left
45	18	165	48	17.63	right	37	21	169	63	22.06	right
48	18	144	68	32.79	right	41	28	178	100	31.56	right
49	23	155	45	18.73	right	42	21	165	63	23.14	right
50	18	155	59	24.56	right	43	21	175	80	26.12	right
51	18	167	63	22.59	right	46	21	188	84	23.77	right
52	20	162	54	20.58	right	47	18	175	80	26.12	left
53	23	179	60	18.73	right	60	21	178	73	23.04	right
54	18	165	70	25.71	right	71	18	173	59	19.71	right
55	20	161	62	23.92	right	75	19	162	86	32.77	right
56	19	176	72	23.24	right	87	18	185	76	22.21	right
57	17	170	61	21.11	right						
58	18	158	52	20.83	right						
59	18	170	68	23.53	right						
61	18	167	74	26.53	right						
62	17	177	69	22.02	right						
63	18	160	58	22.66	right						
64	18	165	49	18.00	right						
65	19	174	58	19.16	right						
66	18	162	50	19.05	right						
67	18	174	59	19.49	right						
68	20	174	57	18.83	right						
69	19	161	65	25.08	right						
70	17	178	68	21.46	right						
72	20	158	60	24.03	right						
73	18	162	57	21.72	right						
74	19	171	61	20.86	right						
76	19	164	61	22.68	right						
77	19	170	63	21.80	right						
78	18	150	46	20.44	right						
79	19	168	77	27.28	right						
80	19	155	70	29.14	right						
81	18	165	59	21.67	left						
82	17	168	59	20.90	right						
83	18	178	61	19.25	right						
84	20	165	63	23.14	right						
85	19	174	64	21.14	right						
86	18	168	59	20.90	right						
88	19	168	57	20.20	right						
89	21	165	54	19.83	right						
90	32	165	58	21.30	right						

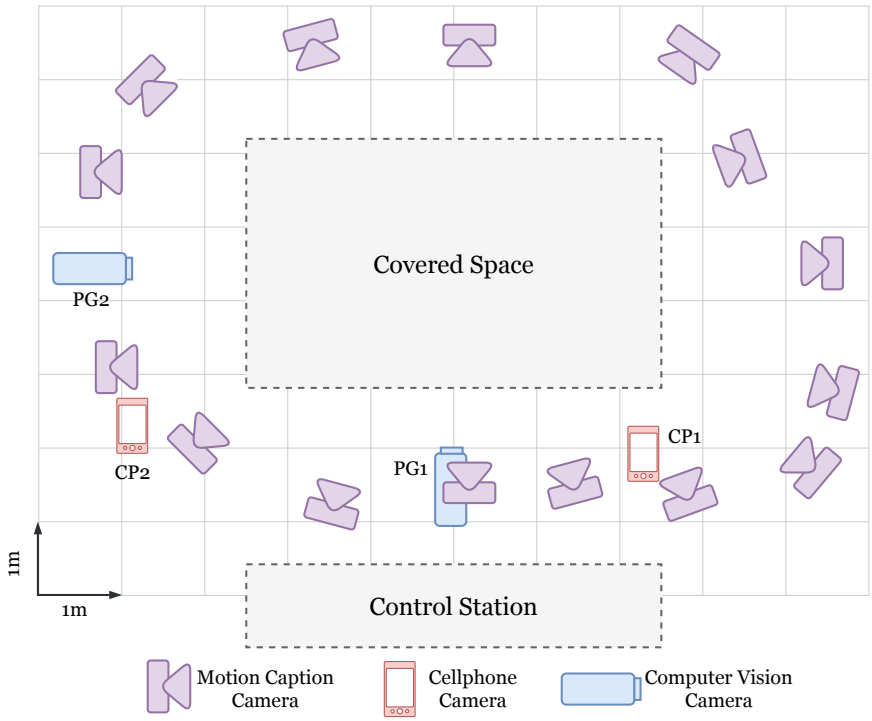


Figure 3.1: Top view sketch of the capture room set-up. The position of the video cameras and motion capture cameras were arranged to cover a space of approximately 3 by 5 meters.

cm diameter with a frame rate of 120 frames per second (fps). The Qualisys Track Manager (QTM) software was used for the acquisition of the optical motion capture data and for setting the synchronization triggering a signal that was sent to the Grasshopper video cameras that were connected to the motion capture system.

Video Cameras. Video data were collected using two hand-held smartphone cameras and two stationary computer vision cameras. For the hand-held cameras, the rear camera of the iPhone 7 (Apple Inc., USA, https://www.apple.com/) was used. The camera has a resolution of 1920×1080 pixels and contains the Sony IExmor RS, CMOS sensor. The video data was recorded with a frame rate of 30 fps. As computer vision cameras, we used RGB Grasshopper2 cameras (FLIR Systems Inc., USA, https://www.flir.com/) with a resolution of 800×600 pixels, 72 dpi, 24-bit depth and Sony ICX285 CCD sensors. The recording with these cameras was also done with a frame rate of 30 fps. The FlyCapture software provided by FLIR Inc. was used for setting up the cameras' acquisition features and for processing the synchronization triggering signal coming from the motion capture system. We also integrated the MATLAB Image Acquisition Toolbox as it

Figure 3.2: Example pictures of one female and one male actor wearing the IMU suits used for the capture rounds S1, S2, I1, and I2.

supports the Grasshopper computer vision cameras and provides blocks and functionalities such as hardware triggering, configuring acquisition parameters and recorded data format, and previewing the recorded data.

Inertial Measurement Unit Sensors. The Noitom Neuron Edition V2[1] was used which comes as a bodysuit attached with 17 IMU sensors (Figures 3.2 and 3.3). Each sensor is composed of a 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer working at 120 Hz. In addition to the acceleration data, the IMU suit provides computed 3D displacements, velocity, quaternions, and rotational velocity for each joint (all represented in an initial global coordinate system). The IMU sensors' dynamic range, accelerometer range, and gyroscope range are 360 deg, ± 16 g, and ± 2000 deg/s, respectively. The static error of the sensors is less than 1 deg for all roll, pitch, and yaw angles. The AXIS NEURON software provided by Noitom LTD was used for setting the acquisition features, calibration of the sensors, data capturing, validation of the recorded data, and exporting the files to different formats.

3.3.3 Data Collection

Participants went through five data capturing sequences. The sequences differed in the hardware systems used to capture the motions, in participants' clothing (minimal, or normal), and whether or not there was a rest pose between successive motions. An overview of the different capture

[1]Noitom LTD, China, https://www.noitom.com/

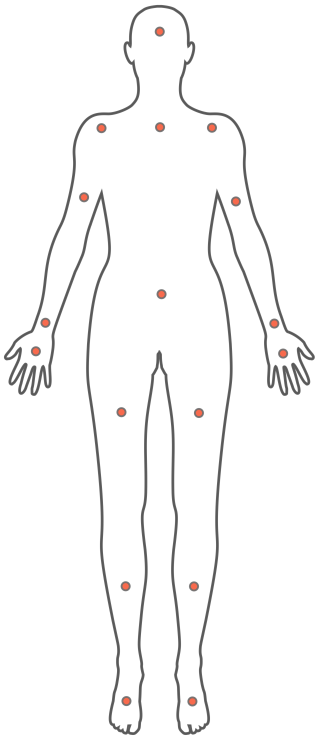


Figure 3.3: Placement of IMU sensors on the body.

rounds is provided in Table 3.2. In each sequence, participants performed the same predefined set of 20 movements in a randomized order and additionally one self-chosen movement, based on verbal instructions by the experimenter. The movements included everyday actions and sports movements: (1) walking, (2) jogging, (3) running in place, (4) side gallop, (5) crawling, (6) vertical jumping, (7) jumping jacks, (8) kicking, (9) stretching, (10) crossing arms, (11) sitting down on a chair, (11) crossing legs while sitting, (13) pointing, (14) clapping hands, (15) scratching one's head, (16) throwing and catching, (17) waving, (18) taking a picture, (19) talking on the phone, (20) checking one's watch, (21) performing a self-chosen movement. To allow for more variation in performed movements in each action class, we did not set any constraints on how exactly each action had to be performed.

Data capture sequence "F". This sequence was captured using the 67 MoSh motion capture marker layout [129]. Subjects wore tight-fitting minimal clothing in order to minimize marker movement relative to the body. The markers were attached to the actors' skin and clothes using double-sided tape. In addition to the motion capturing, video material was recorded using two

Table 3.2: Overview of the five different capture rounds. F = full motion capture markerset, S = sparse motion capture markerset + IMU, I = IMU; *1 = with rest A-pose, *2 = without rest A-pose.

Data Capture Sequence	F	S1	S2	I1	I2
Motion capture markerset	67	12	12	–	–
Video capture	yes	yes	yes	yes	yes
IMU	no	yes	yes	yes	yes
A-pose between motions	yes	yes	no	yes	no
Actor clothing	minimal	normal clothing	normal clothing	normal clothing	normal clothing
Length (min per person)	~ 2.7	~ 2.7	~ 1.7	~ 2.7	~ 1.7

stationary Grasshopper cameras and the rear cameras of two hand-held iPhones 7. For details on the synchronization of the motion capture system and the stationary cameras, see Synchronization Section. Participants performed the actions separated by a rest A-pose. The motivation for this capture round was to obtain accurate full skeletal (pose) information and frame-by-frame body shape parameters without any artifacts imposed by clothing. Therefore, this round is suitable for 2D or 3D pose estimation and tracking, and 3D shape reconstruction.

Data capture sequences "S1" and "S2". For these two sequences, subjects wore the IMU body-suit and a reduced optical motion markerset layout of 12 motion capture markers that were attached to their body (4 markers placed on the head, 2 on each ankle and 2 on each wrist). In addition, the actions were recorded using synchronized computer vision cameras (see Synchronization section), and iPhone 7 rear cameras. In "S1", there was a rest A-pose between the actions, whereas in "S2", there was a natural transition between the performed actions. The reason for choosing a small motion capture markerset was that it provides sparse, but accurate data for some of the main end-effectors including the head, wrists, and ankles, and at the same time allows participants to wear natural clothing.

Data capture sequences "I1" and "I2". These two sequences were captured with participants wearing the IMU suit under their normal clothing. Additionally, video material was recorded using the hand-held iPhone 7 and stationary Grasshopper video cameras. Motions in "I1" are separated by a rest A-pose, whereas there is a natural transition between the actions in "I2".

3.3.4 Preprocessing

3.3.4.1 Motion Capture Data

A cubic polynomial gap filling was automatically done in the QTM software for gaps of less than or equal to 5 frames. The trajectories were then labelled manually using the integrated trajectory identification tool. The resulting labelled trajectories were then exported to a C3D format.

3.3.4.2 Video Data

Each data capture sequence was recorded in one piece, without stopping the recording between the different actions. Therefore, the recorded sequences by the computer vision cameras were manually trimmed into individual single actions and the time-stamps (frame numbers) of the start and end of each action were exported. Based on the time-stamps, the corresponding synchronized motion capture and IMU data were also trimmed into the same individual single actions.

3.3.4.3 IMU Data

The original IMU data stored in calculation file format (`.calc`) were reorganized and converted into MATLAB `.mat` files to get the data in a more readable structure.

3.3.5 Calibration

3.3.5.1 Motion Capture Cameras

The calibration of the motion capture cameras was done before each recording session following the measurement protocol in the Qualisys Track Manager software [130]. The software allows computing the orientation and position of each camera in order to track and perform calculations on the 2D data for conversion into 3D data. The average residual error of the calibration was kept below 0.8 mm and the calibration was repeated if this threshold was not met.

3.3.5.2 Video Cameras

To compute the intrinsic parameters of the Grasshopper computer vision cameras and lens distortion parameters, the MATLAB Single Camera Calibrator [131, 132, 133] was used, where focal length

($F \in \mathbb{R}^2$), optical center ($C \in \mathbb{R}^2$), skew coefficient ($S \in \mathbb{R}$), and radial distortion ($D \in \mathbb{R}^2$) are estimated for each camera. The average re-projection error was kept to less than 0.2 pixels, and the calibration was repeated for higher error values. No calibration was performed for the iPhone cameras.

3.3.5.3 IMU Device

Model Posture Calibration: Before starting each session, a four-step calibration process was required to calibrate the actor's posture. The four-step calibration process is performed by the actor posing in a steady pose, A pose, T pose, and S pose.

Neuron Calibration: IMU sensors might accumulate some calculation errors over time. This usually causes posture computation problems such as drifting. Therefore, each individual IMU sensor should be calibrated after some time of usage. However, to make sure that recordings were accurate enough, we calibrated the sensors before collecting data from each subject following the Noitom Axis Neuron user manual [134].

3.3.5.4 Motion Capture and Video Cross-Calibration

To cross-calibrate the motion capture system with the two Grasshopper computer vision cameras, the location of world points was aligned onto the camera coordinates. For that, the extrinsic parameters which represent the rotation $R \in SO(3)$ and translation $T \in \mathbb{R}^3$ from the motion capture system's coordinate system (world coordinates) to the camera coordinates were estimated using the semi-automated method proposed by Sigal et al. [124]. The trajectory of a single moving marker was recorded by the synchronized motion capture and video cameras for > 2000 frames. Given the recorded 3D positions of the marker in motion capture coordinates as world points and the 2D positions of the marker in the camera frame as image points, the problem of finding the best 2D projection can be formulated as a Perspective-n-Point (PnP) problem where the Perspective-Three-Point (P3P) algorithm [135] is used to minimize the re-projection error as follows:

$$\min_{R,T} \sum_{n=1}^{N} \left\| P_{2D}[n] - f\left(P_{3D}; R, T, K\right)[n] \right\|^2, \tag{3.1}$$

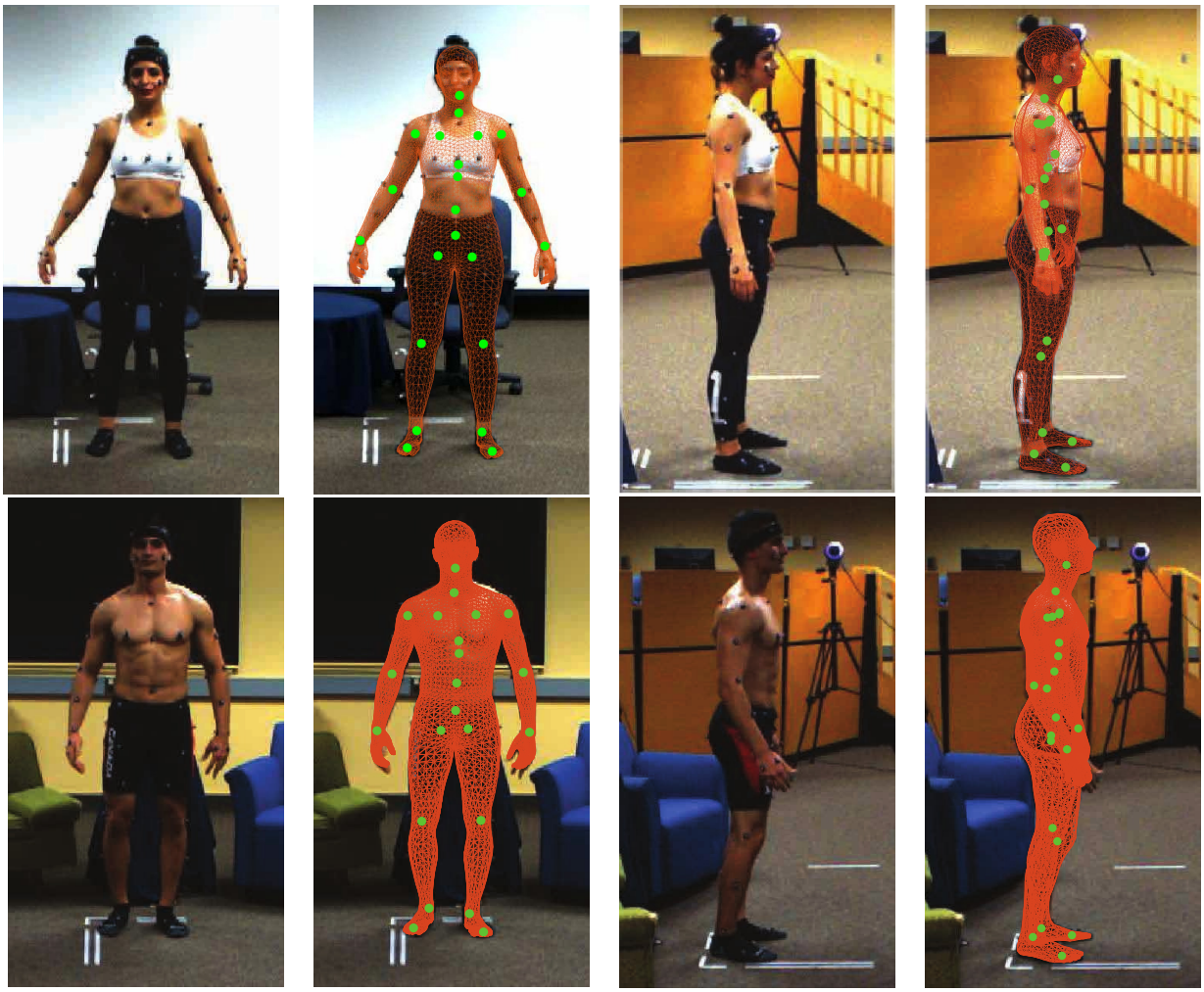


Figure 3.4: Front and side view of aligned video frame, joint locations, and estimated body mesh (computed by MoSh++) for one female and male participant.

where n is the frame number, $N > 2000$ is the total number of recorded frames, f is the projection function and K is the set of camera intrinsic and lens distortion parameters. The 2D position of the single marker was located using a Hough circle transform [136] and double-checked manually frame-by-frame.

To validate the computed extrinsic parameters, the parameters were evaluated in a separate single marker capture session. The average re-projection RMS error on this test run was around 0.8 cm. Synchronization and calibration were additionally validated by careful visual inspection of the accuracy of overlaid joint and mesh positions on the video data for a random selection of multiple rounds. The examples shown in Figure 3.4 are representative for the quality of the whole database.

3.3.6 Synchronization

3.3.6.1 Motion Capture and Video Data

For the data capture sequences "F", "S1", and "S2", the motion capture system and the cameras had to be time-synchronized. In our setup, the video cameras were triggered by the synchronization signal from the QTM software of the motion capture system through ethernet. Due to the frame rate limits in the video cameras, the synchronization frequency was divided by 4, which reduced the video capture frame rate to 30 fps. The iPhone cameras were not synchronized with the motion capture cameras.

3.3.6.2 Motion Capture and IMU Data

To use both IMU and motion capture data in the sequences "S1" and "S2" in a data fusion scenario, these modalities needed to be synchronized in frames. To this end, the cross-correlation between the z-axis location of ankles was used which was pre-computed in these two modalities. The two coordinate systems were not aligned, however, the differences between the orientation of the two z axes are negligible: the z axis of the IMU coordinate system is oriented towards gravity, while z axis in motion capture coordinate system is perpendicular to the floor. Because the motion capture system was synchronized with the video cameras, we additionally obtained synchronized IMU and video data.

Suppose $p_z^j[n]$ and $\tilde{p}_z^j[n]$ are the z component of tracked position of joint j at time-step n recovered by the motion capture and IMU systems, respectively (we are using the 3D positions provided by the IMU software instead of double-integrating over accelerations). The synchronization parameters, temporal scale α and temporal shift τ, are found by maximizing:

$$\max_{\alpha,\tau} \sum_{n=-\infty}^{\infty} p_z^j[n]\tilde{p}_z^j[\alpha n + \tau], \tag{3.2}$$

where the integral is the cross-correlation between $p_z^j[n]$ and scaled version of $\tilde{p}_z^j[n]$. The optimal parameters, by which the highest peak in cross-correlation is achieved, were found using an exhaustive search for $0.9 \leq \alpha \leq 1.1$ (search step-size $= 0.001$) and $-200 \leq \tau \leq 200$. The second term in the summation in Eq. 3.2 was evaluated using spline interpolation. We found $\alpha = 1$ for all

samples meaning that there was no scaling. To ensure that the optimized parameters were robust, we normalized the resulted cross-correlation (Eq. 3.2) to the maximum of 1 and only accepted those where the distance between the first and second peak was higher than 0.3. Only in 3 out of all "S1" and "S2" rounds the parameters got rejected and the synchronization was repeated. Finally, we did a visual inspection of all accepted samples.

3.3.7 Skeleton and Body Shape Extraction

The motion capture data collected in *"F"* was processed using two different pipelines to compute the skeleton: Visual3D [18] (C-Motion Inc., USA, https://c-motion.com/) and MoSh++ [122, 129] (https://amass.is.tue.mpg.de/). The data collected in *"S"* was processed using Visual3D and the same formulas for computing head, wrists, and ankles joint positions. Example images of one female and male participant in rest A-pose with overlaid joint locations and mesh are shown in Figure 3.4.

3.3.7.1 Visual3D software

Visual3D is a biomechanics analysis software for 3D motion capture data [18]. In our Visual3D pipeline, the pelvic segment was created using CODA [137] and the hip joint positions were estimated using Bell and Brand's hip joint center regression [138, 139]. The upper body parts were estimated using the Golem/Plug-in Gait Upper Extremity model [140]. The resulting skeleton at each frame is represented by 20 joints in two different formats: 1) in local joint transformations, that is, the orientation and translation of each joint relative to the coordinate system of its parent joint in the kinematic tree, and 2) as global 3D joint locations.

3.3.7.2 MoSh++

MoSh++ is an approach which estimates the body shape, pose, and soft tissue deformation directly from motion capture data [122]. Body shape and pose are represented using the rigged body model SMPL [121] where the pose is defined by joint angles and shape is specified by shape blend shapes. MoSh++ achieves lower errors compared to the original MoSh framework [129], which used the SCAPE body model [141]. It uses a generative inference approach whereby the SMPL body shape and pose parameters are optimized to minimize reconstruction errors. The skeletal joint locations are computed using a linear regression function of mesh vertices. The estimated SMPL body is

extended by adding dynamic blend shapes using the dynamic shape space of DMPL to simulate soft tissue deformations. Each frame in the "MoSh-ed" representation includes 16 SMPL shape coefficients, 8 DMPL dynamic soft-tissue coefficients, and 66 SMPL pose coefficients as joint angles (21 joints + 1 root). MoSh-ed data of our motion capture recordings was computed in collaboration with the authors of AMASS [122].

The main difference between MoSh++ and Visual3D is that the models are optimized for different applications. MoSh++ is a better choice for character animation, and pose estimation and tracking, whereas Visual3D is preferred for gait analyses and biomechanics. MoSh++, on the one hand, can provide an estimate of joint transformations for all joints even if marker occlusion occurs. However, the estimated joint locations can be noisy when occlusions occur and the error may propagate to other joints. This is because MoSh++ uses distributed information by regressing from the inferred body mesh to the skeleton joints. For character animations, however, precise joint locations are often not important. For gait analysis and biomechanics applications, on the other hand, an accurate estimation of joint locations is crucial. Visual3D achieves this by doing the computations locally where each joint location is computed only from the surrounding markers. The only drawback of Visual3D representation compared to MoSh++ is that the joints cannot be computed at all if one of contributing markers is occluded. In the database, we indicated the time-stamps of the frames where such occlusions occurred.

3.4 Data Records

The file structures of the raw and processed data which are provided in the MoVi Dataverse repository [142], with naming conventions and detailed descriptions are provided in Table 3.3.

Table 3.3: Naming conventions and structure of all files available from the database. ⟨ID⟩ ∈{1,2,...,90} indicates the subject number, ⟨round⟩ ∈{F,S1,S2,I1,I2} the data collection round, and ⟨camera⟩ ∈{PG1,PG2,CP1,CP2} the camera type where PG indicates the computer vision cameras and CP the cellphone cameras.

Data Type	File Name	Description
Video Data	⟨round⟩_⟨camera⟩_Subject_⟨ID⟩.⟨format⟩	avi video data from the computer vision cameras (PG1, PG2) for rounds F, S1, and S2, and mp4 video data from the cellphone cameras (CP1, CP2) and all rounds (F, S1, S2, I1, and I2) and all subjects (1-90). Note that we provide code to trim the video sequences to single motion clips for round F.
Camera Parameters	cameraParams_⟨camera⟩.⟨format⟩	Contains the camera intrinsic calibration data for camera PG1 and PG2 in .mat, .npz, and .pkl formats. These parameters are fixed for the whole dataset.
	Extrinsics_⟨camera⟩.⟨format⟩	Contains the camera extrinsics parameters for camera PG1 and PG2 (rotation matrix and translation vector) in .mat, .npz, and .pkl formats.
Motion Capture Data	F_amass_Subject_⟨ID⟩.mat	Contains the full markerset motion capture data (round F) processed by MoSh++ in the AMASS project and augmented with 3D joint positions and metadata for each subject (1-90). All files are compressed and stored as F_AMASS.tar. The original npz files and the rendered animation files are available at https://amass.is.tue.mpg.de/. Note that the we provide code to trim the motion capture sequences to single motion clips.
	F_v3d_Subject_⟨ID⟩.mat	Contains the full markerset motion capture data (round F) processed by Visual3D and augmented with metadata for each subject (1-90). All files are compressed and stored as F_Subjects_⟨ID⟩_⟨ID⟩.tar as containers of 45 subjects (e.g., ID 1-45, ID 46-90).
	S_v3d_Subject_⟨ID⟩.mat	Contains the motion capture data from rounds S1 and S2 processed by Visual3D and augmented with metadata. All files are compressed and stored as S_V3D.tar.
IMU Data	imu_Subject_⟨ID⟩.mat	Contains the processed IMU calculation files augmented with metadata. Each file contains the data collected in all rounds (S1, S2, I1, I2). The files are compressed as IMUmatlab_Subject_⟨ID⟩_⟨ID⟩.tar containers of 15 subjects (e.g., ID 1-15, ID 16-30 etc).
	imu_Subject_⟨ID⟩.bvh	Contains IMU in .bvh format. Each file contains the data collected in all rounds (S1, S2, I1, I2). The files are compressed as IMUbvh_Subject_⟨ID⟩_⟨ID⟩.tar containers of 15 subjects (e.g., ID 1-15, ID 16-30 etc).

3.4.1 Raw data

Raw video data from the computer vision cameras is provided as `.avi` video files to avoid any artifacts added by compression methods. Raw motion capture data stored as `.qtm` files that are only readable by the QTM software and `.c3d` and raw IMU data stored in `.xml` and `.calc` file formats are not included in the MoVi database. However, these files can be provided upon request.

3.4.2 Processed data

The processed full markerset motion capture data (capture round "F") is provided in two different versions based on the post-processing pipeline (MoSh++/AMASS and Visual3D). We provide joint angles and 3D joint locations computed by both pipelines along with the associated kinematic tree, information about the occlusions and optical marker data. Both versions are provided as `.mat` format for each subject. The `.mat` file also contains body pose-independent shape parameters provided by the MoSh++ pipeline as SMPL blend shape coefficients [121]. Given pose-independent shape parameters and joint angles, corrective pose-dependent shape parameters and the resulting surface mesh represented as frame-by-frame 3D vertices can be computed. Due to the reduced markerset, the motion capture data collected in rounds "S1" and "S2" were only processed using the Visual3D pipeline for extracting the head, wrists, and ankles joint positions provided as `.mat` files. Synchronized IMU data were computed by processing the `.calc` files and converting them to `.mat` format which provides raw acceleration data, displacement, velocity, quaternions, and angular velocity. The `.bvh` files generated by the IMU software are also provided in the repository.

3.5 Applications

By the time of publishing, MoVi dataset was the only synchronized and cross-calibrated video, motion capture, and IMU dataset that provides accurate 3D body shape and pose. Importantly, by using different combinations of hardware systems to record the same actors and motions, the dataset provides overlapping information that can facilitate training models for body shape reconstruction, and pose estimation and tracking from video data.

For body shape reconstruction tasks, our dataset provides 3D body shape based on the SMPL

model which provides not only pose-independent shape parameters, but also pose-dependent shape parameters, and therefore allows for more accurate shape representation.

For body pose estimation tasks, our dataset contains two formats of body pose representations based on motion capture data, Visual3D [18] and SMPL/MoSh++ [121, 122]. Visual3D is a biomechanical model that provides accurate estimation of joint locations if no marker occlusions occur, and is therefore suitable for motion modelling and gait analysis. MoSh++ provides an estimate of all joint locations (although noisy when occlusions occur), and is therefore more suitable for pose estimation and tracking tasks. In addition to the capture round *F* with the full optical marker set, we used a sparse set of optical markers in rounds *S1* and *S2* to reduce the visual artifacts in the video material. The sparse marker set still provides ground truth 3D position of the main joints while still featuring natural clothing. MoVi also provides challenging action types that are useful for training robust pose estimation models, such as cross-legged sitting and crawling, but that are not commonly seen in other datasets with ground truth 3D pose.

The large number of 90 individual actors who performed the same set of actions, provides high diversity across performers in terms of action type, action execution, style, and modalities (video, motion capture, and IMU) which are important factors for research on action recognition (see e.g., [143] who used our dataset for action recognition). This is also important for frameworks for designing character animation pipelines that focus on modelling the natural stochasticity and diversity of the movements (e.g, [30]).

3.6 Code availability

The custom MATLAB and Python scripts for processing the data are provided on the following Github repository:
https://github.com/saeed1262/MoVi-Toolbox. The repository contains all the necessary tools for file reading, conversion, processing, and visualization. An additional tutorial is provided on how to use the dataset.

3.7 Conclusion

The MoVi dataset includes five data subsets that were recorded using synchronized video, optical motion capture and IMU hardware systems to provide partially overlapping information across the different subsets. It features the same 60 female and 30 male actors who repeat the same set of 21 everyday motions and sports movements in each data subset. In total, MoVi contains 9 hours of optical motion capture data, 17 hours of video data recorded from 4 different points of view with both hand-held and stationary cameras, and 6.6 hours of IMU data. To our knowledge, our dataset is the largest dataset in terms of the number of subjects and performed motions, and the first dataset with the synchronized pose, pose-dependent and pose-independent body shape, and video recordings.

Chapter 4

Auto-labelling of Markers in Optical Motion Capture by Permutation Learning

The work in this chapter has been published previously as the following:

Saeed Ghorbani, Ali Etemad, and Nikolaus F. Troje, "Auto-labelling of Markers in Optical Motion Capture by Permutation Learning", in *Computer Graphics International Conference*, 2019

and has been updated and extended here for the purposes of this document.

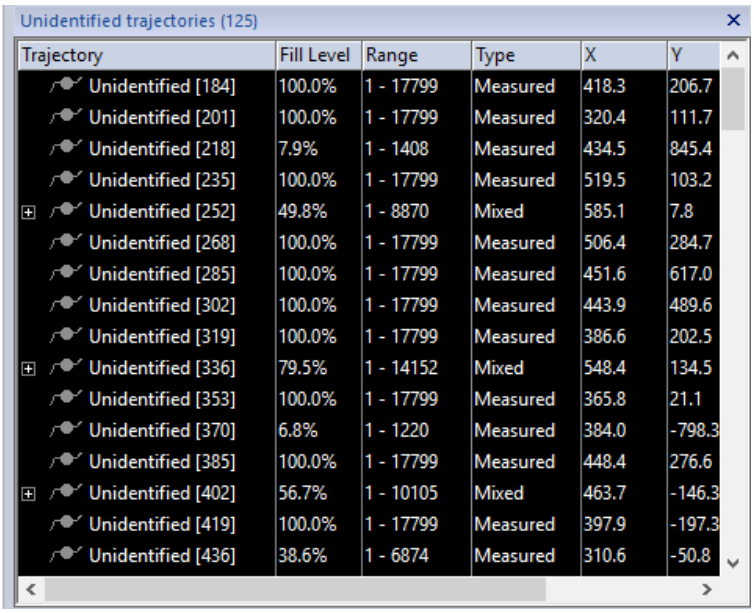
4.1 Preface

In recent years optical marker-based motion capture has been a vital tool for capturing biological motions to be used in applications such as motion and behavioural analysis, animation, and biomechanics. Collecting motion data using optical motion capture systems involves some manual steps before and after each capture session. *Labelling*, that is, assigning optical markers to the pre-defined positions on the body, is one of the main time-consuming and labour intensive post-processing parts of current motion capture pipelines. It requires technicians to spend hours on manual labelling of the trajectories for each individual capture session. The amount of needed time and attention can also increase exponentially for complex movements which makes labelling one of the main friction points in optical motion capture pipelines. In this chapter, we present a framework for automatic labelling to alleviate this problem in the current pipelines.

We formulated the problem of marker labelling as a ranking process in which markers shuffled by an unknown permutation matrix are sorted to recover the correct order. Our proposed framework for automatic marker labelling first estimates a permutation matrix for each individual frame using a differentiable permutation learning model and then utilizes temporal consistency to identify and correct remaining labelling errors. Experiments conducted on the test data show the effectiveness of our framework.

4.2 Background

In optical motion capture pipelines, after each capture session a set of recorded 3D trajectories are listed in a random order. Each trajectory represents the motion of a single visible marker in terms of its 3D position over time (see Figure 4.1). The first step in this state is labelling these unlabelled (a.k.a unidentified) trajectories, which is assigning each individual trajectory to a specific body location. This process can be very time-consuming and therefore a friction point for many motion capture systems. Furthermore, this process is susceptible to user errors and more challenging when markers are occluded (for simplicity we use the term *occlusion* for any type of missing marker in the data) over time, hence splitting the motion of these markers into multiple trajectories and represented with different IDs in the motion capture software. Commercial motion



Trajectory	Fill Level	Range	Type	X	Y
Unidentified [184]	100.0%	1 - 17799	Measured	418.3	206.7
Unidentified [201]	100.0%	1 - 17799	Measured	320.4	111.7
Unidentified [218]	7.9%	1 - 1408	Measured	434.5	845.4
Unidentified [235]	100.0%	1 - 17799	Measured	519.5	103.2
Unidentified [252]	49.8%	1 - 8870	Mixed	585.1	7.8
Unidentified [268]	100.0%	1 - 17799	Measured	506.4	284.7
Unidentified [285]	100.0%	1 - 17799	Measured	451.6	617.0
Unidentified [302]	100.0%	1 - 17799	Measured	443.9	489.6
Unidentified [319]	100.0%	1 - 17799	Measured	386.6	202.5
Unidentified [336]	79.5%	1 - 14152	Mixed	548.4	134.5
Unidentified [353]	100.0%	1 - 17799	Measured	365.8	21.1
Unidentified [370]	6.8%	1 - 1220	Measured	384.0	-798.3
Unidentified [385]	100.0%	1 - 17799	Measured	448.4	276.6
Unidentified [402]	56.7%	1 - 10105	Mixed	463.7	-146.3
Unidentified [419]	100.0%	1 - 17799	Measured	397.9	-197.3
Unidentified [436]	38.6%	1 - 6874	Measured	310.6	-50.8

Figure 4.1: A sample of list of unlabelled trajectories after a capture sessions

capture software usually provide tools to reduce the amount of manual work in this step. These tools are usually based on a model that can learn the geometry of the participant's body and applies it to label subsequent measurements. However, they typically require manual initialization where one or more motion capture sessions have to be labelled manually for each participant.

Typically, motion capture labelling comprises two main steps: the initialization step, where the initial correspondences are established for the first frame, and the tracking step which can be defined as keeping track of labelled markers in the presence of occlusion, ghost markers, and noise. A number of approaches have been introduced to address the tracking of manually initialized motion capture data. Holden [24] proposed a deep de-noising feed-forward network that outputs the joint transformations directly from corrupted markers. First, the training data were synthesized using a large motion capture dataset, skinning the skeletal motion data using linear blend skinning, reconstructing virtual markers locations, and finally corrupting marker data by removing or adding noise similar to those present in the real motion capture data. Then, a de-noising feed-forward neural network was trained on the synthetic data to learn the mapping from corrupted marker data to the joints transformation. Herda et al. [144] proposed an approach to increase the robustness of optical markers specifically during visibility constraints and occlusions by using the kinematics information provided by a generic human skeletal model. Yu et al. [145] proposed an online motion

capture approach for multiple subjects which also recovers missing markers. They used the standard deviation of the distance between each pair of markers to cluster the markers into a number of rigid bodies. Having fitted rigid bodies, they labelled the markers using a structural model for each rigid body and a motion model for each marker. Loper [19] proposed a marker placement refinement by optimizing the parameters of a statistical body model in a generative inference process. In recent work, Ghorbani and Black [146] proposed a neural network framework called *SOMA* which takes point clouds of unlabelled optical motion capture markers and outputs the labels assigned to each point using self-attention layers [147] for learning multi-scale 3D point features, and an optimal transport layer [148] to constrain the assignment space and handle unmatched cases. They provide an end-to-end framework by fitting a SMPL-X [149] body model using MoSh [19].

Another group of approaches attempts to minimize user intervention by automating the initialization step as well. Holzreiter [21] trained a neural network to estimate the positions of sorted markers from a shuffled set. Labelling of the markers was carried out by pairing up the estimated marker locations with the shuffled set using the nearest neighbour search. Meyer et al. [150] estimated the skeletal configuration by least-squares optimization and exploited the skeletal model to automatically label the markers. They applied the Hungarian method for optimal assignment of observation to markers while requiring each subject to go into a T-pose to initialize the skeletal tracker. Schubert et al. [151] improved their approach by designing a pose-free initialization step, searching over a large database of poses. Finally, Han et al. [23] and Maycock et al. [22] proposed auto-labelling approaches specifically designed for hands. Han et al. [23] proposed a technique to label the hand markers by formulating the task as a keypoint regression problem. They rendered the marker locations as a depth image and fed it into a convolutional neural network which outputs the labelled estimated 3D locations of the markers. Then, they used a bipartite matching method to map the labels onto the actual 3D markers. Maycock et al [22] used inverse kinematics to filter out unrealistic postures and computed the assignment between model nodes and 3D points using an adapted version of the Hungarian method. Although their approach was focused on hands, it can be applied to human body motion.

In this chapter, we are proposing an end-to-end, data-driven approach for automatic labelling which does not require a manual initialization. We formulated the labelling at each frame as a permutation estimation where shuffled markers are ranked based on a pre-defined order. Then,

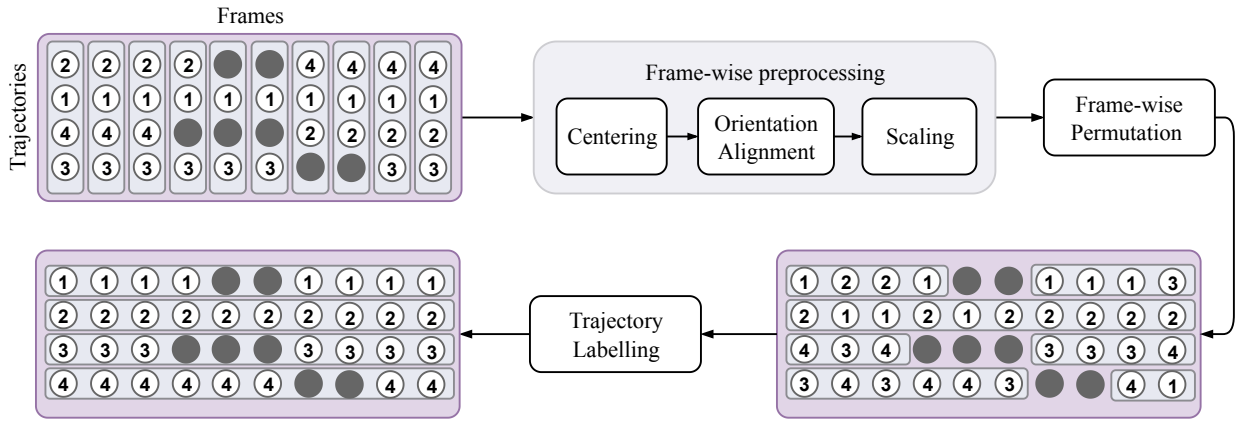


Figure 4.2: An overview of our proposed framework: the input to our system is a sequence of unlabelled (shuffled) 3D trajectories. After the preprocessing stage, the label for each marker is estimated by applying permutation learning to each individual frame. The resulting labelled frames are then concatenated to form trajectories again. These trajectories are then used as input to the trajectory labelling stage where a temporal consistency constraint is used to correct mislabelled markers

in a trajectory labelling step, temporal consistency is used to correct mislabelled markers. Our framework can be reliably run in real-time which potentially results in faster, cheaper, and more consistent data processing in motion capture pipelines.

4.3 Proposed Framework

Figure 4.2 illustrates the block diagram of our main framework. The first stage of our framework is a preprocessing step which is applied to individual frames, making the array of markers invariant to spatial transformations. For the next stage, we propose a data-driven approach that avoids the need for manual initialization by formulating automatic labelling as a permutation learning problem for each individual frame. Towards this end, we present a permutation learning model which can be trained end-to-end using a gradient-based optimizer. We exploited the idea of relaxing our objective function by using doubly-stochastic matrices as a continuous approximation of permutation matrices [152]. During the running phase, each individual frame is automatically labelled using the proposed permutation learning model. The result is a sequence of individually labelled frames where each trajectory might be assigned to multiple labels over time. We then evaluate temporal consistency in the resulting trajectories and use it to identify and correct the labelling errors that occurred during the previous stage. To correct the inconsistencies in each marker trajectory, a score

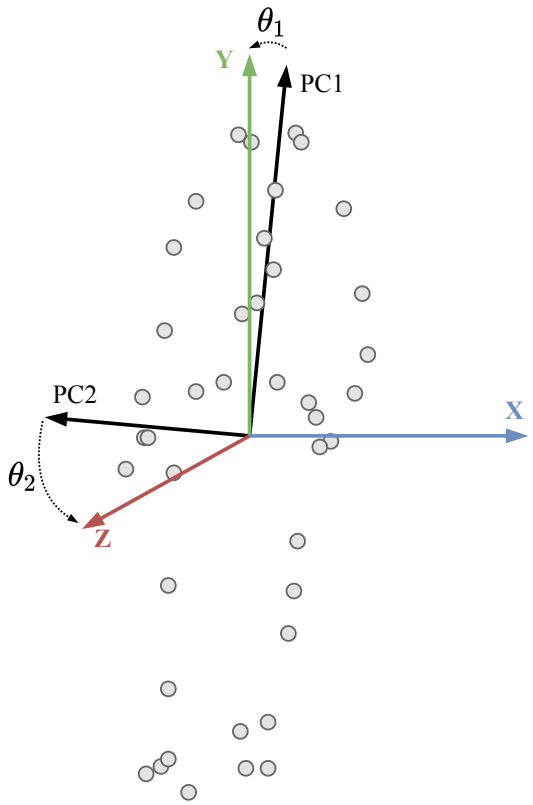
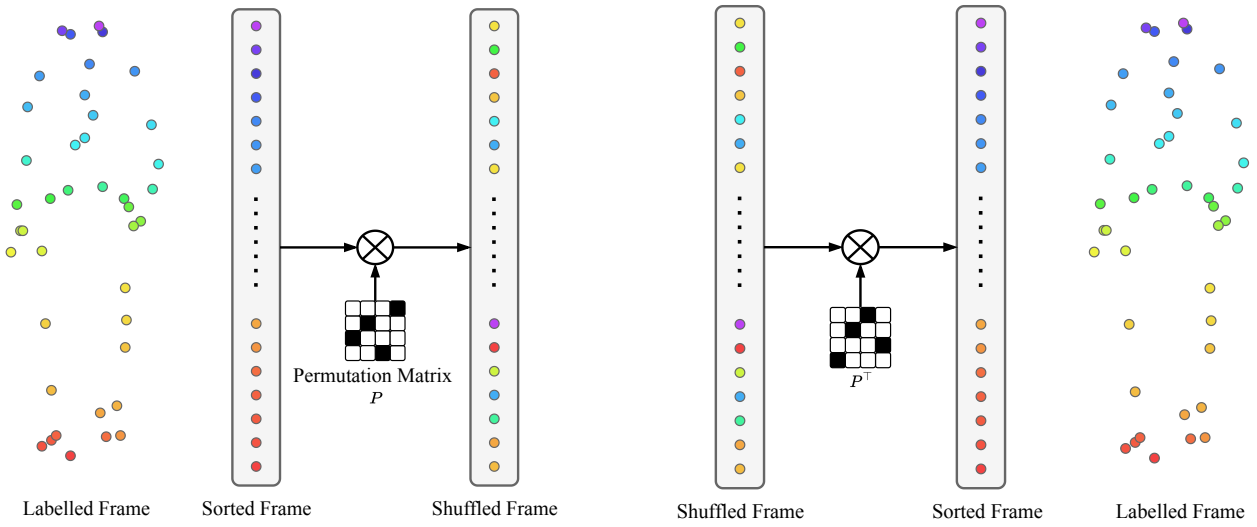


Figure 4.3: We make the data invariant to the orientation of the subject by aligning the first and second PCA components of the cloud of the markers to the y and z axes, respectively.

is computed for each candidate label using a confidence-based score function. Then, the label with the highest score will be assigned to the marker trajectory in a winner-takes-all scheme.

4.3.1 Data Preprocessing

Prior to applying our permutation learning model, we ensure that the input data are invariant to translation, orientation, and the size of the subjects. We first calculated the centroid of the array of markers for each frame and then subtracted it from the marker locations to make each frame invariant to translation. To make the data invariant to the orientation of the subject, we applied principal component analysis (PCA) to the cloud of the markers. We first aligned the direction with the largest principal component with the y-axis. Then, the second principal component was aligned to the z-axis to make the poses invariant to the rotations around y-axis (Figure 4.3). Finally, the size of the subject was normalized by scaling the values between 0 and 1 independently for each of



(a) Unlabelled markers

(b) Labelled markers

Figure 4.4: Formulating the labelling task as a permutation learning model. Unlabelled markers can be considered as shuffled version of sorted labels by a permutation matrix (figure a). Labelling task is then can be defined as finding this permutation matrix given the 3D location of the markers and applying its transpose on the shuffled data to sort the makers based on a predefined order of labels (figure b).

the three spatial dimensions.

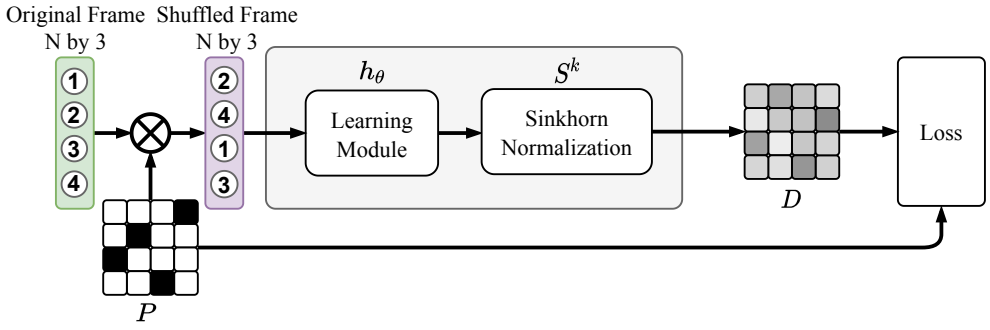
4.3.2 Permutation Learning Model

The motion capture data at each frame can be represented by the 3D positions of N markers utilized in the recording process. Labelling in each frame is defined as assigning the 3D positions of these markers to specific, fixed body locations. This process can be formulated as permuting a set of shuffled 3D elements to match a pre-defined order. Let us define a labelled frame as $X = [m_1,\ m_2, \ldots,\ m_N]^\top \in \mathbb{R}^{N\times 3}$, to be an ordered array of N markers, where m_i represents the 3D position of the i_{th} marker. Then, a shuffled frame $\tilde{X} = PX$ can be considered as a permuted version of X where the markers are permuted by a permutation matrix $P \in \{0,\ 1\}^{N\times N}$ (Figure 4.4a). Hence, given a shuffled frame, the original frame can be recovered by multiplying the shuffled version with the inverse of the respective permutation matrix (Figure 4.4b). It should be noted that for a permutation matrix P, $P^\top = P^{-1}$. Our goal in this step is to design a trainable parameterized model $f_\theta : \mathcal{X}^N \rightarrow \mathcal{S}_N$ which takes a shuffled frame $\tilde{X}$ as input and estimates the permutation matrix P that was originally applied to the frame. Then, having the permutation

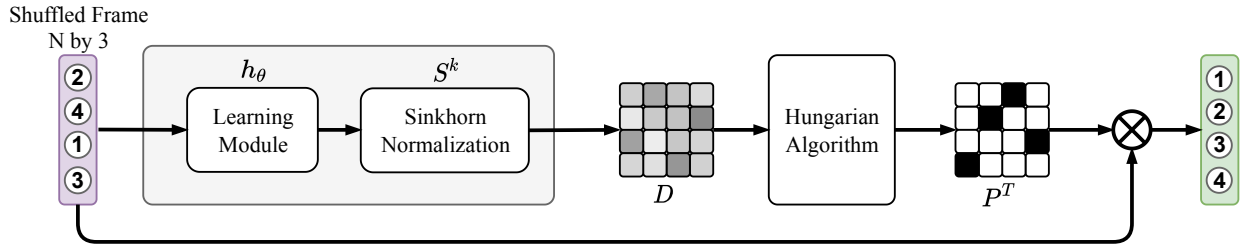
matrix P we can recover the sorted frame X, where $X = P^\top \tilde{X}$.

The main difficulty in training a permutation learning model using backpropagation is that the space of permutations is not continuous which prohibits computation of the gradient of the objective function with respect to the learning parameters since it is not differentiable in terms of the permutation matrix elements. To address this problem, Adams et al. [152] proposed the idea of utilizing a continuous distribution over assignments by using doubly-stochastic matrices as differentiable relaxations of permutation matrices. This approach has been successfully exploited in other applications [153, 154]. A DSM is a square matrix populated with non-negative real numbers where each of the rows and columns sums to 1. All $N \times N$ DSM matrices form a convex polytope known as the *Birkhoff Polytope* $\mathcal{B}_N$ lying on a $(N-1)^2$ dimensional space where the set of all $N \times N$ permutation matrices are located exactly on the vertices of this polytope [155]. Therefore, DSMs can be considered as continuous relaxations of corresponding permutation matrices. We can interpret each column i of a DSM as a probability distribution over labels to be assigned to the i_{th} marker. Also, all rows summing to 1 ensures the inherent structure of permutation matrices. Accordingly, instead of mapping directly from 3D positions to the permutation matrices, we propose to learn a model $g_\theta : \mathcal{X}^N \rightarrow \mathcal{W}_N$, where $\mathcal{W}_N$ is the set of all $N \times N$ DSMs. That way, computing the permutation matrix from the DSM simply becomes a bipartite matching problem that can be solved $\mathcal{O}\left(N^3\right)$.

Our permutation learning model is composed of two main modules: the learning module and the Sinkhorn normalization (see Figure 4.5). The learning module $h_\theta : \mathcal{X}^N \rightarrow \mathbb{R}_+^{N \times N}$ is the parameterized component of our model, which learns the feature representation of the data structure from the available poses, taking the 3D positions in each frame and outputting an unconstrained square matrix. We implement this module as a feed-forward deep residual neural network [156]. The last dense layer consists of $N \times N$ nodes with a sigmoid activation outputting an $N \times N$ non-negative matrix. One naïve approach would be to treat the problem as a multi-class, multi-label classification task with N^2 classes. However, this approach would ignore the inherent structure of permutation matrices with the possibility of resulting in impossible solutions. To enforce the optimizer to avoid these erroneous solutions we need our model to output a DSM which replicates the inherent structure of permutation matrices. An effective way to convert any unconstrained non-negative matrix to a DSM is using an iterative operator known as Sinkhorn normalization [157, 152].



(a) Learning Phase



(b) Runtime Phase

Figure 4.5: An overview of our permutation learning model: In (a), the learning phase is depicted, where the parameters of the learning module are optimized to minimize the multi-class, multi-label cross-entropy loss. In (b), the running phase is illustrated, where the Hungarian algorithm is applied to the outputted DSM to estimate the optimal permutation matrix which is then applied to the shuffled labels to sort them to the predefined order.

This method normalizes the rows and columns iteratively where each pair of iterations is defined as:

$$S^i(M) = \begin{cases} M & \text{if } i = 0 \\ \mathcal{T}_R(\mathcal{T}_C(S^{i-1}(M))) & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\mathcal{T}_R^{i,j}(M) = \frac{M_{i,j}}{\sum_k M_{i,k}}$ and $\mathcal{T}_C^{i,j}(M) = \frac{M_{i,j}}{\sum_k M_{k,j}}$ are the row and column-wise normalization operators, respectively. The Sinkhorn normalization operator is defined as $S^\infty : \mathbb{R}_+^{N\times N} \to \mathcal{W}_N$ where the output converges to a DSM. We approximate the Sinkhorn normalization by an incomplete version of it with $k < \infty$ pairs of iteration. Knight [158] proved that to reach ϵ-near doubly stochasticity, $\mathcal{O}(B|\log\epsilon|)$ iterative steps are needed when all of the matrix elements are bounded in $[0, B]$. The defined Sinkhorn normalization function is differentiable and the gradients of the learning objective can be computed by backpropagating through the unfolded sequence of row and column-wise normalizations, unconstrained matrix, and finally learning module parameters. The

gradient of row and column-wise normalization operators can be computed as follows:

$$\frac{\partial}{\partial M_{i,j}}\mathcal{L}\left(\mathcal{T}_{\mathrm{R}}(M)\right)=\sum_{j'=1}^{N}\left[\frac{\delta(j-j')}{\sum_{j''=1}^{N}M_{i,j''}}-\frac{M_{i,j'}}{\left(\sum_{j''=1}^{N}M_{i,j''}\right)^{2}}\right]\frac{\partial\mathcal{L}\left(\mathcal{T}_{\mathrm{R}}(M)\right)}{\partial\mathcal{T}_{\mathrm{R}}(M)_{i,j'}}, \tag{4.2}$$

$$\frac{\partial}{\partial M_{i,j}}\mathcal{L}\left(\mathcal{T}_{\mathrm{C}}(M)\right)=\sum_{i'=1}^{N}\left[\frac{\delta(i-i')}{\sum_{i''=1}^{N}M_{i'',j}}-\frac{M_{i',j}}{\left(\sum_{i''=1}^{N}M_{i'',j}\right)^{2}}\right]\frac{\partial\mathcal{L}\left(\mathcal{T}_{\mathrm{C}}(M)\right)}{\partial\mathcal{T}_{\mathrm{C}}(M)_{i',j}}. \tag{4.3}$$

During training, the parameters of learning modules are optimized by minimizing the distance between the computed DSM and the ground truth permutation matrix.

$$\mathcal{L}=\underset{\theta}{\text{minimize}}\sum_{(X,P)\in\mathcal{D}}d\left(P,S^{k}(h_{\theta}(\tilde{X}))\right), \tag{4.4}$$

where d measures the distance between the computed DSM and the ground truth permutation matrix P. In our work, we formulated the distance as a multi-class, multi-label cross-entropy loss. S^k is a Sinkhorn normalization layer with k iteration steps.

During the running phase, a single permutation matrix P' must be predicted by finding the closest polytope vertex to the doubly-stochastic matrix D produced by the model. This can be formulated as a bipartite matching problem where the cost matrix is $C = 1 - D$. The optimal permutation matrix can be found by minimizing the assignment cost.

As a result, we use the Hungarian algorithm [159] over the cost matrix to find the optimal solution to the matching problem. Hungarian algorithm can solve the assignment problem in polynomial time ($O\left(n^3\right)$ for Jonker–Volgenant algorithm [160]). Finally, each individual frame is sorted (labelled) by the transpose of the corresponding estimated permutation. The learning and runtime phases are illustrated in Figure 4.5.

4.3.3 Trajectory Labelling

After applying our permutation learning model to all the frames in the entire sequence, we have a sequence of individually labelled frames ordered in time. However, the integration of sequences of individual marker locations into trajectories that extend over time, which is already conducted by the motion capture system during data collection, and the expectation that labels should remain

constant during the motion trajectory, can be used to enforce temporal consistency. Each trajectory can be defined as the sequence of tracked marker locations which ends with a gap or when recording stops. Therefore, in each motion sample, the movement of each marker might be presented in multiple trajectories over time. We can exploit the temporal consistency of each trajectory to correct the wrong predictions for each marker during the trajectory. One naïve idea is to assign each trajectory to the label with the highest number of votes in the assignment predictions for the corresponding marker. However, there are situations where a label has been assigned to a marker with the highest number of times but with low confidence. Thus, we propose a winner-takes-all approach where a score is computed for each label which has been assigned at least once to the marker in the trajectory. Then the winner with the highest score will be assigned to the corresponding trajectory. The score for label i assigned to the query marker j is computed as follows:

$$S_i = |T_i|^q \left(\sum_{t \in T_i} |c_{i,j}^{(t)}|^p \right)^{\frac{1}{p}}, \tag{4.5}$$

where T_i is the set of frame indices at which label i has been assigned to the query marker during its trajectory. $|T_i|$ is the cardinality of T_i. p and q are hyperparameters which are chosen during the validation step. The second term in right-hand side for $p = 0$ is defined as $|T_i|$. $c_{i,j}^{(t)}$ represents the degree of confidence for assigning the label i to the marker j at frame t. By formulating the score function in this way, we are taking into account both the number of frames (first term) and the L^p-norm of the degree of confidences (second term) when each label is assigned to the query marker. The details of $c_{i,j}^{(t)}$ formulation are discussed in the next section.

4.3.3.1 Degree of Confidence

During the running phase, each column of the outputted DSM matrix represents a distribution over labels for the corresponding marker which can be interpreted as the model's belief in each label. When the model is confident in labelling all markers, the estimated DSM matrix is close to the true permutation matrix on the polytope surface and all of these distributions peak at the true label. On the other hand, when a marker is hard to label, the corresponding distribution might not have a sharp peak at the true label. We measure the degree of confidence for each predicted label i assigned to marker j at frame t as the distance between the model's belief for label i and

the highest belief:

$$c_{i,j}^{(t)} = D_{i,j}^{(t)} - \max_{\substack{1 \le k \le N \\ k \neq i}} D_{k,j}^{(t)}, \tag{4.6}$$

where $D^{(t)}$ is the DSM matrix produced by Sinkhorn normalization at frame t. We also tried other formulations such as the model's belief for label i:

$$c_{i,j}^{(t)} = D_{i,j}^{(t)}, \tag{4.7}$$

or the distance between the model's belief for label i and the average of the beliefs for the other labels:

$$c_{i,j}^{(t)} = D_{i,j}^{(t)} - \frac{1}{N-1} \sum_{\substack{1 \le k \le N \\ k \neq i}} D_{k,j}^{(t)}, \tag{4.8}$$

but got the best results when using the Eq. 4.6 formulation. Note that the defined $c_{i,j}^{(t)}$ for Eq. 4.6 and Eq. 4.8 can be negative. Therefore, we used a min-max normalized version of it ($c_{i,t} \leftarrow (c_{i,t}+1)/2$) to map its range to between 0 and 1.

4.4 Experiments and Evaluations

4.4.1 Data

To evaluate our method we used two different motion capture datasets. Our original evaluation was performed using a subset of BMLrub dataset [48] recorded from 115 individuals at 120 frames per second using Vicon motion capture system. The subset includes four types of actions, namely walking, jogging, sitting, and jumping, where each recording contains the 3D trajectories of 41 markers. Some actions where recorded multiple times from the same individual. On average, we used 11.5 sequences per subject for a total of 1329 sequences. The total number of frames in our dataset is around 630k frames. Data from 69 individuals were used for training, while the data from 46 different individuals were held out for testing and validation (23 each).

We also did a post-hoc evaluation on the MoVi dataset [161] which was recorded from 90 individuals each performing 21 types of actions (One sequence for each action). The motion capture was done using Qualisys motion capture system at 120 frames per second with the marker layout

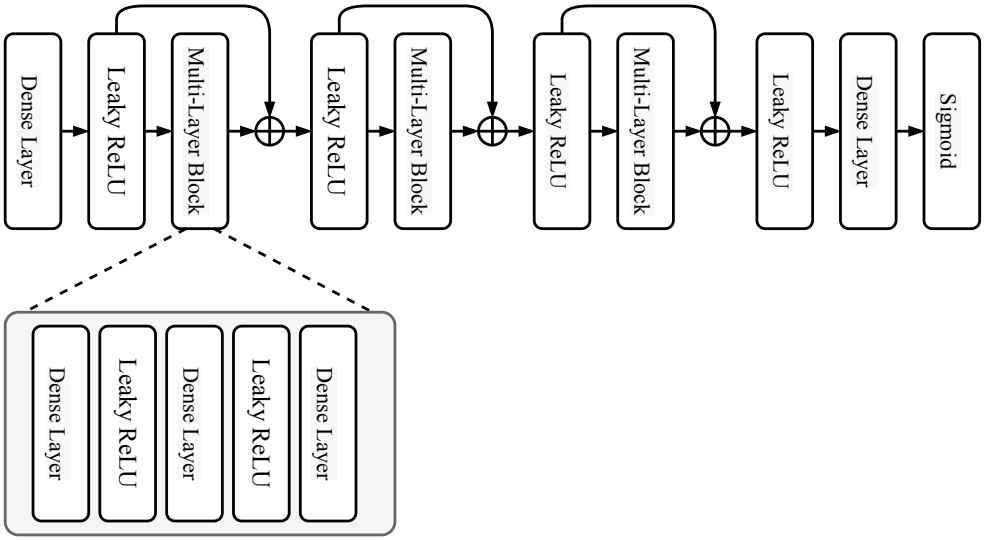


Figure 4.6: Schematic diagram of the feed-forward deep residual neural network used for permutation learning.

suggested in MoSh [19] including 61 makers. The total number of frames in the dataset is around 1.27M frames. We split the data by subjects to $46, 22, 22$ subjects for training, evaluation, and test, respectively.

4.4.2 Training

The feed-forward residual network implemented as our learning module was designed with three residual blocks where each block contains three dense layers followed by a Leaky ReLU activation (see Figure 4.6). The residual connections showed a smoothing behavior on our optimization landscape. Hyper-parameters of the network were chosen using a random search scheme [162].

The original training set was constructed by applying 16 random permutations on each of the training frames resulting in around 6 million shuffled training frames for bmlRUB and around 10 million shuffled frames for MoVi. Then, to augment the training data with the occlusions, up to 5 markers in each generated shuffled frame were randomly occluded by replacing the 3D position values with the center location $(0.5, 0.5, 0.5)$. Both the numbers of occlusions and the index of occluded markers were drawn from uniform distributions.

For training the model, we used an Adam optimizer with batches of size 32. The learning rate was scheduled using a reduce-on-plateau scheme where it was initially set to 5×10^{-4} for bmlRUB

and 2.5×10^{-4} for MoVi. Then the learning rate was reduced with a factor of 2 when the loss of validation set stopped decreasing with a patience parameter of 5 epochs. Our model was trained for 100 epochs for bmlRUB and 140 epochs for MoVi using a multi-class, multi-label cross-entropy loss function. The number of Sinkhorn iterations was set to 5 since for each additional iteration the improvement in performance was very small while the running time was increased linearly. By performing these 5 Sinkhorn iterations on our unconstrained matrix, the sum of squared distances between 1 and the row- and column-wise sums of the resulting matrix were less than 10^{-17} for both bmlRUB and MoVi datasets.

4.4.3 Permutation Learning Model Evaluation

To evaluate our permutation learning model, we synthesized an evaluation set by applying 16 random permutations followed by randomly introducing 1 to 5 occlusions into each frame of the test set until we have an equal number of samples for each number-of-occlusions. Table 4.1a shows the performance of our model in different setups and compares them with the initialization steps proposed by Holzreiter et al. [21] and Maycock et al. [22]. The performance is measured as the average labelling accuracy where the accuracy for each frame is defined as the number of markers correctly labelled divided by the total number of markers. The first and second rows in table 4.1a show the average accuracy results when the model was trained on the original training set and the occlusion augmented set, respectively. As anticipated, introducing occlusions into the training data improves the results for the test frames with occluded markers, which is usual in real scenarios. On the other hand, when the model is trained without occlusions the performance on occluded frames significantly decreases.

To evaluate the influence of Sinkhorn normalization, we replaced the Sinkhorn layer with a Softmax function over the rows of outputted DSM matrix and trained the parameters from scratch. The results for these setups are illustrated in the third row of table 4.1a. Without Sinkhorn normalization, the output matrix ignores the inherent structure of permutation matrices resulting in a drop in the labelling performance.

Having defined the degree of confidence, there is the option to only assign a label to a marker if the corresponding degree of confidence is higher than a threshold. Otherwise, the marker is left unlabelled. This allows the model to set a trade-off between precision (the fraction of correctly

(a) BMLrub

Method	Number of Occlusions					
	0	1	2	3	4	5
Ours + Occs	97.11%	**96.56%**	**96.13%**	**95.87%**	**95.75%**	**94.9%**
Ours + No Occs	**98.72%**	94.41%	92.15%	88.75%	86.54%	85.0%
Ours w/o SN	94.03%	91.12%	88.1%	84.27%	81.62%	77.78%
Maycock et al.	83.18%	79.35%	76.44%	74.91%	71.17%	65.83%
Holzreiter et al.	88.16%	79.0%	72.42%	67.16%	61.31%	52.1%

(b) MoVi

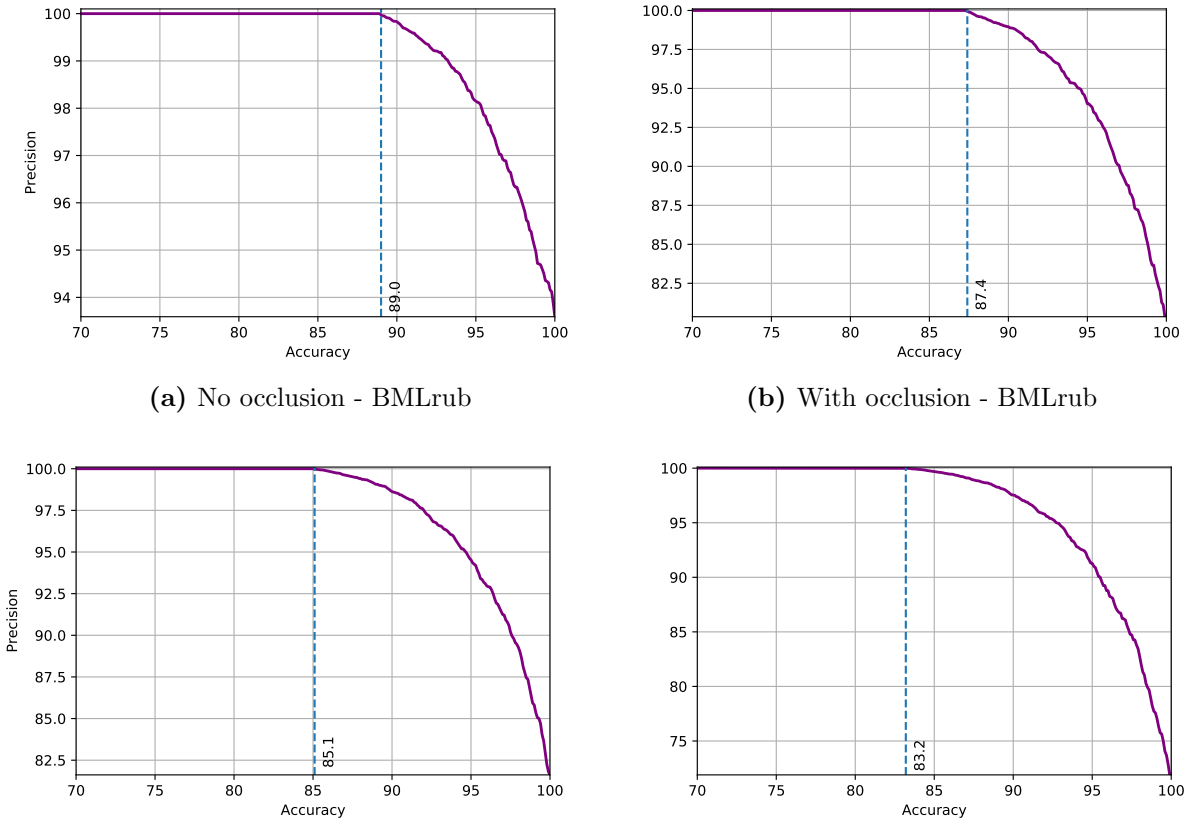
Method	Number of Occlusions					
	0	1	2	3	4	5
Ours + Occs	93.21%	**93.05%**	**92.87%**	**92.54%**	**92.41%**	**92.19%**
Ours + No Occs	**94.32%**	92.17%	88.42%	84.45%	83.14%	82.51%
Ours w/o SN	90.13%	88.47%	84.23%	81.12%	78.51%	74.91%
Maycock et al.	80.71%	76.11%	72.68%	71.39%	67.77%	61.92%
Holzreiter et al.	84.29%	75.3%	71.18%	62.08%	55.74%	48.13%

Table 4.1: A comparison of performance of different models in labelling a single test frame as an initialization step in the presence of a varying number of occlusions in the test frames (show in each column). The first and second rows illustrates the performance when the model is trained with and without occlusions, respectively. Third row, shows the results when the Sinkhorn layer is replaced by a Softmax function. It can be seen that the model trained on occlusion augmented training set outperforms the rest when having occlusions in the frames.

labelled markers over labelled markers) and accuracy (the fraction of correctly labelled markers overall markers). Figure 4.7 shows the accuracy-precision curves where the threshold has been gradually decreased. It can be seen that a high proportion of the markers can be labelled with no error (89% and 85.1% when there is no occlusion and 87.3% and 83.2% with an average of 2.5 markers occluded in each frame for BMLrub and MoVi, respectively) and leaving less than 12.6% markers for BMLrub and 16.8% markers for MoVi to be labelled manually.

4.4.4 Trajectory Labelling Evaluation

So far, we have looked at frames individually. In the next stage, we integrate frame-by-frame labelling with information about the temporal order of the frames in an effort to label continuous trajectories. To evaluate the trajectory labelling stage, we used the unlabelled test set and intro-



(a) No occlusion - BMLrub

(b) With occlusion - BMLrub

(c) No occlusion - MoVi

(d) With occlusion - MoVi

Figure 4.7: Plot of accuracy-precision curves for labelling frames without and with occlusions. In the latter case an average of 2.5 markers was missing in each frame.

duced occlusions to each motion sample until we got different occlusion ratios. The occlusions were introduced as gaps into trajectories where the motion of the marker was fragmented into two or more trajectories.

Table 4.2a and 4.2b show the performance of trajectory labelling with different settings of p and q in our scoring function for BMLrub and MoVi marker layouts. When $p = 0$ and $q = 0$, this stage acts as a voting function ($S_i = |T_i|$). Thus, the degree of confidence does not have an influence on the final result. For $p = 1$ and $q = 0$, the scoring function computes the sum of confidences for the frames that the label has been assigned to the marker. Also, when $p = 1$ and $q = -1$, the score for each label is considered as the average of confidences. Therefore, the number of times that a label is assigned to a marker is neutralized by averaging. Best performance in our hyper-parameters search was achieved when $p = 2$ and $q = -1/2$. In this case, the influence of N_i on the score is less

(a) BMLrub

Method	Occlusions Ratio					
	0	2%	4%	6%	8%	10%
No trajectory labelling	97.13%	96.55%	96.17%	95.91%	95.68%	94.82%
$p = 0$, $q = 0$	97.52%	96.81%	96.55%	96.09%	96.51%	95.43%
$p = 1$, $q = 0$	98.77%	98.01%	97.78%	97.32%	97.12%	96.85%
$p = 1$, $q = -1$	97.35%	96.51%	96.42%	96.06%	95.89%	94.37%
$p = 2$, $q = -1/2$	**99.85%**	**99.54%**	**99.47%**	**99.25%**	**99.07%**	**98.76%**

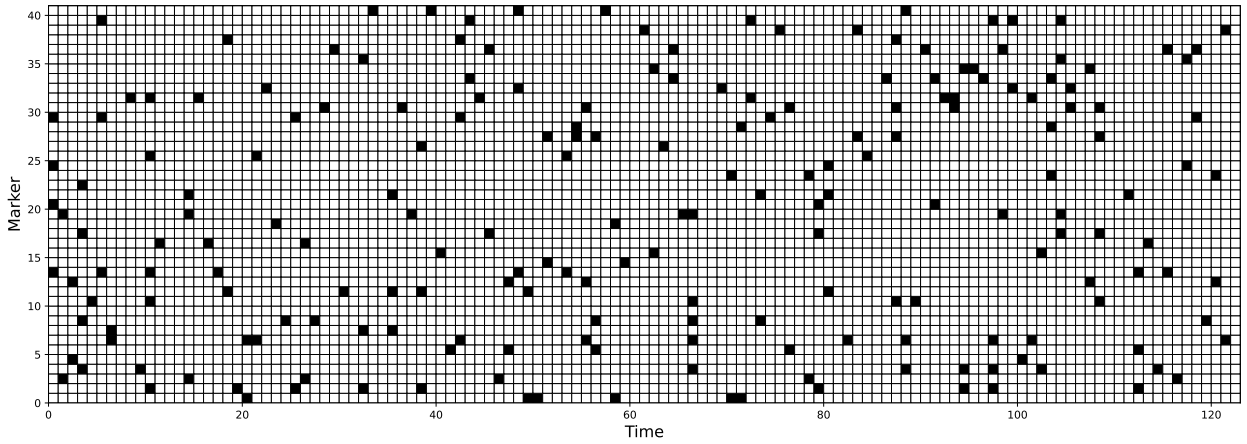
(b) MoVi

Method	Occlusions Ratio					
	0	2%	4%	6%	8%	10%
No trajectory labelling	93.11%	92.77%	92.21%	91.46%	90.73%	90.1%
$p = 0$, $q = 0$	93.43%	92.9%	92.42%	91.96%	91.26%	90.89%
$p = 1$, $q = 0$	93.92%	93.25%	92.56%	92.03%	91.48%	91.01%
$p = 1$, $q = -1$	93.26%	92.71%	92.55%	91.83%	91.34%	90.5%
$p = 2$, $q = -1/2$	**95.9%**	**95.52%**	**95.41%**	**95.22%**	**95.02%**	**94.8%**

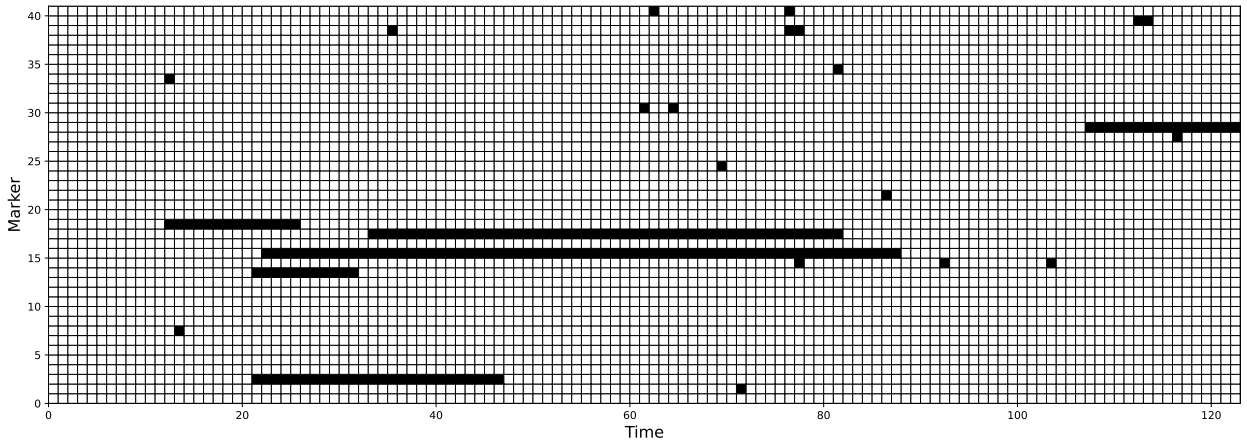
Table 4.2: The performance of trajectory labelling with different settings of p and q in our scoring function.

than when $q = -1$. Also, since p is set to 2, the role of the high degree of confidences is more than other settings.

Note that the occluded markers were drawn from a uniform distribution over all frames and markers which can be considered the worst-case scenario. This is because in the real scenarios the gaps usually tend to be longer than one and therefore there are less number of gaps and longer trajectories for the same occlusion ratios. Figure 4.8 compares the simulated occluded trajectories and a real sample drawn from MoVi dataset with the same occlusion ratio. It can be seen that the average observed trajectory is longer in the real scenarios than in the simulated version.



(a) Simulated occlusions for trajectory labelling evaluation. Equal probability for occlusions is assigned to each marker and each frame. Average gap length = 1.03, Average trajectory length = 21.52



(b) A sample of captured trajectories. Average gap length = 7.18, Average trajectory length = 42.56

Figure 4.8: A comparison between simulated occlusions for our trajectory labelling evaluation and real scenario for the 41 marker layout. In both figures, the occlusion ratio is 4%. In, (a) the markers and frames are occluded with a uniform probability. In (b), we searched for an equal-length window with the same occlusion ratio in the real data. The average trajectory length in simulated occlusions is much higher than the real scenarios.

4.5 Conclusion

In this chapter, we presented a method to address the problem of auto-labelling markers in optical motion capture pipelines. The essence of our approach was to frame the problem of single-frame labelling as a permutation learning task where the ordered set of markers can be recovered by estimating the permutation matrix from a shuffled set of markers. We exploited the idea of using DSMs to represent a distribution over the permutations. Also, we proposed a robust solution to

correct the mislabelled markers by utilizing the temporal information where the label with a higher confidence score is assigned to the whole trajectory. We demonstrated that our method performed with very high accuracy even with only single-frame inputs, and when individual markers were occluded. Furthermore, the trajectory labelling will further improve if longer gap-free trajectories are available. Our method can be considered as both initialization and tracking. Once the model is trained, it easily runs on a medium-power CPU at 120 frames per second and can therefore be used for real-time tracking.

4.6 Future Work

Our model makes fast and robust predictions and is easy to train. However, it should be trained on a training set with the same marker layout. One solution could be to synthesize desired training sets by putting virtual markers on the animated body meshes from labelled data using body and motion animating tools such as [19] and to record the motion of virtual markers.

We have addressed the problem of single-subject marker labelling, but have not considered multi-subject scenarios. Future work could explore using clustering approaches and multi-hypothesis generative approaches to separate the subjects and apply the model to each of them.

Here, we have used a completely data-driven approach to label motion capture trajectories. The model could be further improved by integrating both anthropometric and kinematic information into our method where they can serve as priors that further constrain the model.

Chapter 5

Probabilistic Character Motion Synthesis using a Hierarchical Deep Latent Variable Model

The work in this chapter has been published previously as the following:

Saeed Ghorbani, Calden Wloka, Ali Etemad, Marcus A. Brubaker, and Nikolaus F. Troje, Probabilistic "Character Motion Synthesis using a Hierarchical Deep Latent Variable Model", in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2020

and has been updated and extended here for the purposes of this document.

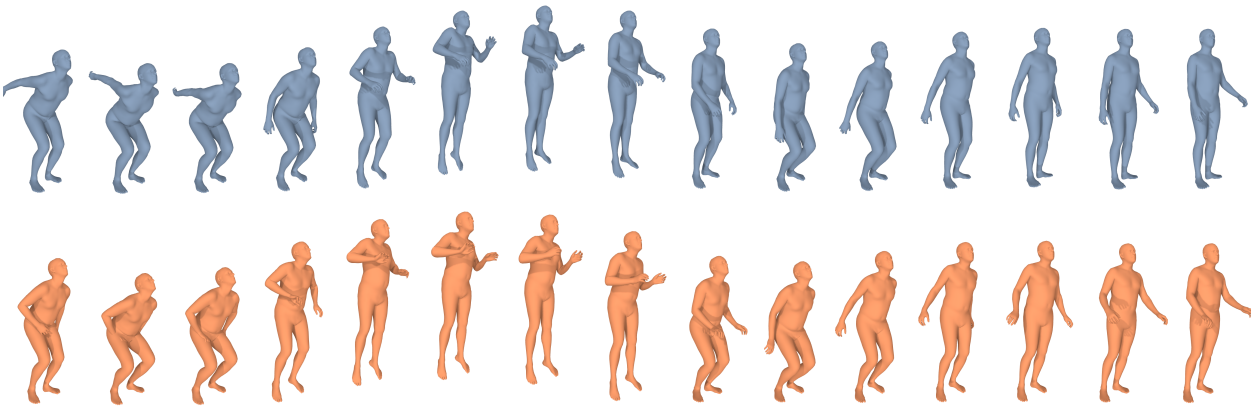


Figure 5.1: Samples of a real motion sequence (blue) and synthesized motion sequence generated by our model (orange)

5.1 Preface

Given the collected dataset and a solution for labelling recorded optical motion capture data presented in the previous chapter, we continue with the development of new data-driven tools using the most recent advances in probabilistic generative models. Such models can take the probabilistic nature of human motion into account, model the full space of plausible human motions, and are able to generate new motion samples in a non-deterministic manner. However, as mentioned earlier, the current approaches for controllable character animation, such as motion matching, which are extensively used in game production, are limited by the amount of used motion capture data. There are two reasons for this: first, the motion capture step is a long, labour-intensive, and expensive process, and second, these approaches must search for a good match at runtime, which can be slow given a large motion capture dataset. The main consequence of this limitation is having a lot of duplicates and repeated patterns in character movements which our visual system can detect after a short period of time. In this chapter, we present a probabilistic framework to generate character animations based on weak control signals such as action type or style type, such that the synthesized motions are realistic while retaining the stochastic nature of the human movement. With minimal supervision, our proposed framework can provide batches of synthesized motion segments with natural within-class variations to be used for data-hungry methods such as motion matching. This alleviates the problem of duplicate and repetitive patterns in such methods. In addition, our model can be used for crowd simulation by generating batches of motion primitives in parallel and

on the fly.

The proposed architecture, which is designed as a hierarchical recurrent model, maps each subsequence of motions into a stochastic latent code using a variational autoencoder extended over the temporal domain. We also propose an objective function that respects the impact of each joint on the pose and compares the joint angles based on angular distance. Finally, we use two novel quantitative protocols and human qualitative assessments to demonstrate the ability of our model to generate convincing and diverse periodic and non-periodic motion sequences without the need for a strong control signal.

5.2 Introduction

An active research area in computer animation is the automatic generation of realistic character animations given a set of control parameters. Recent advances in motion capture technology and deep learning methods have increased interest in data-driven and learnable frameworks for modelling human motion. Most approaches model the motion sequences as a deterministic process, meaning that for a given set of control parameters, only a single, fixed sequence is generated. On the other hand, human motion is stochastic in nature - given the same intention and target, the joints always travel different paths. Hence, deterministic models fail to reflect such diversity, which is essential for generating convincing and realistic character animation. Another challenge in designing a generative model for motion is to enforce desirable motion sequences constrained by weak control signals such as action type. This is due to the fact that deterministic models usually regress to the mean pose in the long run as no strong control signal can be provided, especially for non-periodic movements, to steer the motion and reduce the motion uncertainty over time. Most recent controllable approaches are proposed only for periodic movements with strong control signals such as trajectory characteristics [4, 3, 29], or are limited to short-term predictions for non-periodic movements [2, 1, 67]. Our work addresses these open issues by developing a model for animation synthesis which can be modulated by weak control signals while retaining the desired stochastic characteristics of human motion across both temporal and spatial dimensions.

Our proposed model for character animation synthesis is based on a deep recurrent neural network. We train our recurrent model on a large database of motion capture data such that it can

generate novel, convincing motion samples that imitate the high-level stochastic nature of real data. Furthermore, this semi-supervised framework does not require any manual data preparation such as time-warping or motion clipping which minimizes the amount of manual work in the training and synthesizing processes.

Our framework is designed as a hierarchical recurrent latent variable network that models the spatiotemporal motion data with a two-level hierarchy. The hierarchical structure of the network architecture allows motion sequences to be represented at multiple levels of abstraction and a higher level of the desired variability in the generative process. The inner layer of the proposed architecture is designed as an extension of a variational recurrent neural network [14] which is conditioned on control signals and recursively processes high-level feature vectors (derived from motion subsequences) along with a stochastic latent variable. Defining this latent variable at a high level of abstraction enables the network to model the stochasticity observed in human movement. The inner layer is wrapped by encoder and decoder layers which encode the motion subsequences into feature vectors and decode the generated feature vectors back to motion subsequences.

We also propose a new objective function based on the geodesic distance between the Quaternions representation ground-truth and reconstructed joint angles, which has the following principal advantages: *i*) The geodesic distance better represents the deviation from the desired output than l_p norm losses; *ii*) The influence of different joints in the kinematic tree can be represented by assigning different weights to each joint; *iii*) High-level and semantic information is integrated into the learning process by comparing the ground-truth and reconstructed sequences in the feature space of pre-trained classifiers.

We validated the performance of our model both qualitatively, via human scoring, and quantitatively through a novel evaluation protocol based on the *Inception Score (IS)* [163] and *Fréchet Inception Distance Score (FID)* [164]. These metrics were first used for evaluation in image synthesis and provide a measure not only of the quality of the generated output but also the diversity of output provided. Given the importance of movement variety for character animation synthesis, we have therefore adopted these metrics to provide a more complete evaluation than previously used metrics. The results show that our model effectively learns human motion dynamics and is capable of generating realistic and diverse character animation sequences coherent with control parameters, outperforming all other state-of-the-art models tested.

Our contributions can be summarized as: *i*) we propose a novel hierarchical generative recurrent architecture that effectively learns human motion dynamics and generates realistic character animation sequences coherent with control parameters, *ii*) we present a new objective function based on angular distances and the influence of different components in the kinematic tree, which better represents network error and leads to improved learning, *iii*) we provide a new benchmark and evaluation protocol for character animation synthesis to measure generated sequences' quality and variability.

5.3 System Overview

A visual diagram of our model architecture is given in Figure 5.2. We provide a framework that encapsulates both the hierarchical and the stochastic nature of human motion within a deep hierarchical recurrent architecture. Our model generates motion sequences via a two-level hierarchy. In particular, we model the human motion as a sequence of high-level feature vectors called *Motion Words* where each Motion Word is computed as a function of a sub-sequence of poses. The recurrent processing of motion sequences is thereby applied at word-level rather than at the pose-level.

We leverage an extension of a variational recurrent neural network [14] which contains a variational autoencoder at each time-step conditioned on the control signals. We call the recurrent processing unit a *Motion Cell* which attaches a stochastic latent variable to the observed Motion Words at each time-step. Stochasticity at the Word level enables variability to be represented at a higher level of abstraction (see Section 5.3.2 for details), thereby producing more internally consistent motion sequences. The mapping between Motion Words and the corresponding sub-sequence of poses is performed by f_{enc} and f_{dec}. At each time step, we condition the Motion Cell on the control signals to modulate the motion characteristics and decrease uncertainty due to the multi-modality nature of the motion generation process. In general, any motion-related attribute, static or dynamic, such as style, action type, or motion trajectory, could be used as a control signal. However, in this work, we used a set of holistic attributes consisting of action type and gender.

During training, we integrated individual pre-trained classifiers to the model for each attribute type to infer attributes from the unlabelled input sequence and also to provide additional higher-level learning signals to the objective function. This constrains the model to generate animations

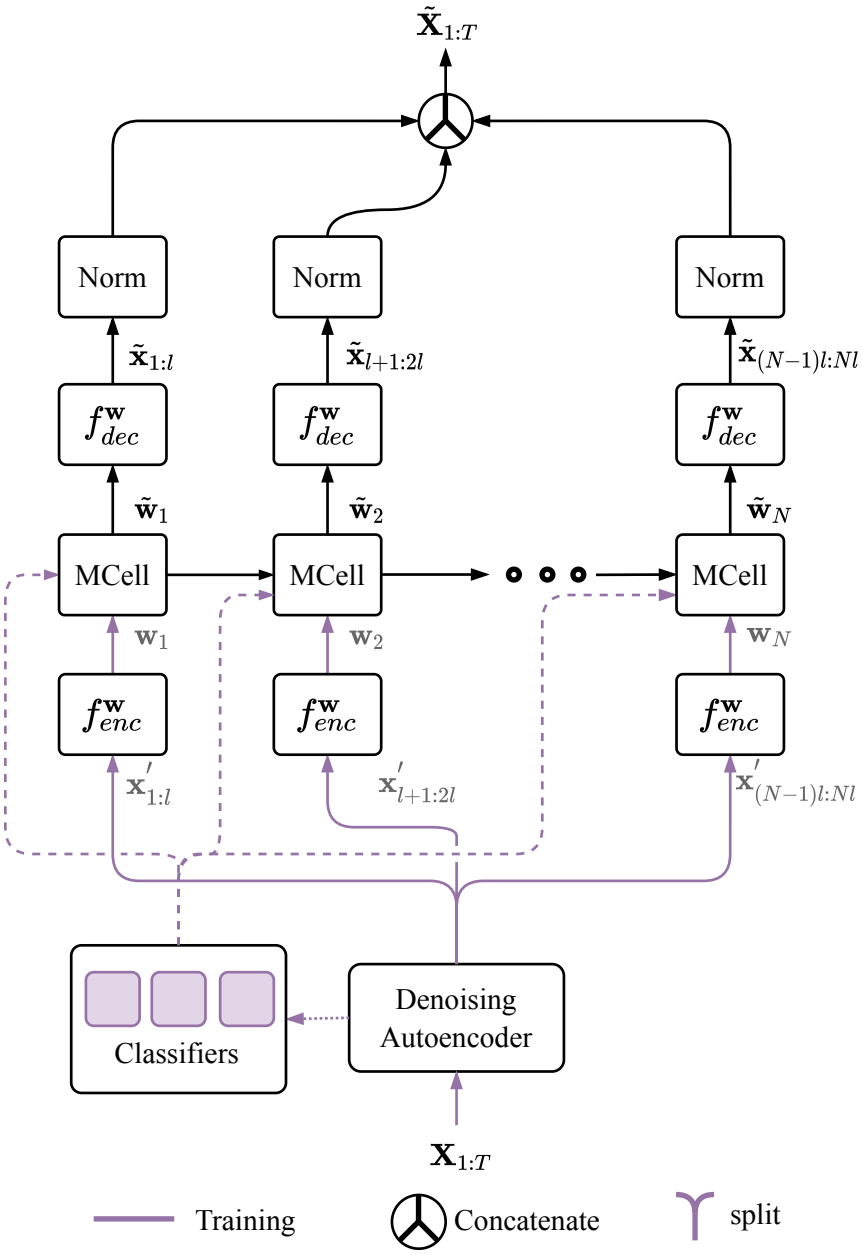


Figure 5.2: An overview of our recurrent model. During training, information is projected into high level feature vectors via f^{w}_{enc}. Motion Cells operate on these feature vectors in latent space to combine information from the preceding sequence with stochastic variability in order to output the next step in the sequence. This output is converted to a joint angle representation via f^{w}_{dec}. Normalization ensures that the representations fall within valid ranges.

that fulfill the semantics defined by the attribute codes (see Section 5.3.4 for details).

Our model uses a joint angle representation to define each pose. We tested the model with three different joint angle representations: axis-angle vectors, quaternions, and rotation matrices. To ensure that the model produces valid rotations for each joint, the estimated rotations in the output

of the hierarchical recurrent neural network are normalized into valid rotations (*Norm* blocks in Figure 5.2). Regardless of the specific joint angle representation used, our model otherwise operates identically from one representation to the next. To have valid rotations represented by quaternions, the magnitude of the quaternions should be one. Therefore, we simply divided each quaternion by its magnitude. When instead using rotation matrices to represent joint angles, we applied the Gram–Schmidt orthonormalization process to the output matrices. No normalization step was applied to the axis-angle vector representation.

5.3.1 Data Preprocessing

The full parameterization input/ouput states of the model at frame i consists of a vector $\mathbf{x}_i = \{\mathbf{j}_i^a, \dot{\mathbf{r}}_i^x, \dot{\mathbf{r}}_i^z, \dot{\mathbf{r}}_i^a, \mathbf{r}_i^y, \mathbf{r}_i^p, \mathbf{r}_i^r, \mathbf{c}_i\} \in \mathbb{R}^{D_s}$ where $\mathbf{j}_i^a \in \mathbb{R}^{kj}$ are the local joint angles for an skeleton with j joints (k is $3, 4$ or 9 for axis-angle vectors, quaternions, and rotation matrices, respectively), $\dot{\mathbf{r}}_i^x \in \mathbb{R}$ is the root transform translational x velocity relative to the forward facing direction, $\dot{\mathbf{r}}_i^z \in \mathbb{R}$ is the root transform translational z velocity relative to the forward facing direction, $\dot{\mathbf{r}}_i^a \in \mathbb{R}$ is angular velocity around the gravitational axis, $\mathbf{r}_i^y \in \mathbb{R}$ is the global root height, $\mathbf{r}_i^p \in \mathbb{R}$ is the pitch and $\mathbf{r}_i^r \in \mathbb{R}$ is the roll relative to the direction where the subject is facing, $\mathbf{c}_i \in \mathbb{R}^6$ is the concatenation of the one hot encodings of the global attributes (The global attributes can be manually annotated or as the output of trained classifiers). During motion synthesis, global translation and orientation can be recovered by integrating velocities over time. The final pose representation consists of a $D_s = j \times k + 12$ dimensional vector. We sub-sampled the motion sequences into 30 frames per second (they were originally recorded in 120 frames per second) and used all four offsets for training.

Our model can be trained by variable-length sequences of inputs. However, to speed up the training process by parallel computing, we set the size of input sequences to a fixed size by clipping the longer sequences and padding zeros to the shorter ones. In our work, we set the length of input sequences to 200 frames (around 6.6 seconds). For synthesis, the length of a generated sequence does not have to be equal to the length of the training sequences, instead our model can generate sequences with arbitrary lengths.

Before feeding to the main model, we apply a pre-trained denoising network to the training sequences to correct possible errors in the training data, such as high-frequency noise resulting from marker occlusions or mislabelled markers in the motion capture process. The denoising network

is implemented as a one-dimensional convolutional denoising autoencoder pretrained on a different subset of data than the one we used for training the main model. Details of the denoising network structure and its training process are given in Section 5.4.2.

5.3.2 Hierarchical Probabilistic Recurrent Network

Our hierarchical recurrent network models a motion sequence of length T with a two-level hierarchy. In the pose-level we have a sequence of poses and in the word-level we have a sequence of Motion Words. Each Motion Word, $\mathbf{w}_n \in \mathbb{R}^{D_w}$, summarizes a sub-sequence of poses using an encoding function

$$\mathbf{w}_n = f_{\text{enc}}^{\mathbf{w}}(\mathbf{X}_{nl:(n+1)l}), \tag{5.1}$$

where $\mathbf{X}_{1:T}$ is the sequence of poses ($\mathbf{X}_t \in \mathbb{R}^{D_p}$), $f_{\text{enc}}^{\mathbf{w}}$ is a non-linear complex encoder such as a fully-connected neural network, and l is the length of each sub-sequence. We define the sequence of Motion Words as an autoregressive model as follows:

$$p\left(\mathbf{w}_1, \ldots, \mathbf{w}_N\right) = p\left(\mathbf{w}_1\right) \prod_{t=2}^{N} p\left(\mathbf{w}_n | \mathbf{w}_{<n}\right), \tag{5.2}$$

where $N = \lfloor T/l \rfloor$ is the number of Motion Words in the sequence. We chose l to be 3 since higher than 3 introduced artifacts to the synthesized motion as discontinuities. The dimension of the Motion Word D_w was set to $32 * 3 = 96$ which is equal to the approximate degrees of freedom in each pose ([12, 165]) times the length of each subsequence. To model the recursive structure of Motion Words we used a variational recurrent neural network [14] extended to condition on the control parameters. The proposed recursive model can be formulated as a recurrent neural network built upon a probabilistic recurrent cell which is structured as a conditional VAE at each time-step. We call these recurrent cells *Motion Cells* (*MCell* blocks in Figure 5.2, see Section 5.3.3 for more details). The combination of a hierarchical structure and probabilistic recurrence allows the model to define a stochastic latent variable at the word-level. Hence, the stochastic behaviour of the generation process is modelled at a deep level using highly abstracted features, allowing variation to be more easily sampled in an internally consistent manner from the learned feature space. We use another fully connected layer to convert the generated Motion Words back to the sub-sequence

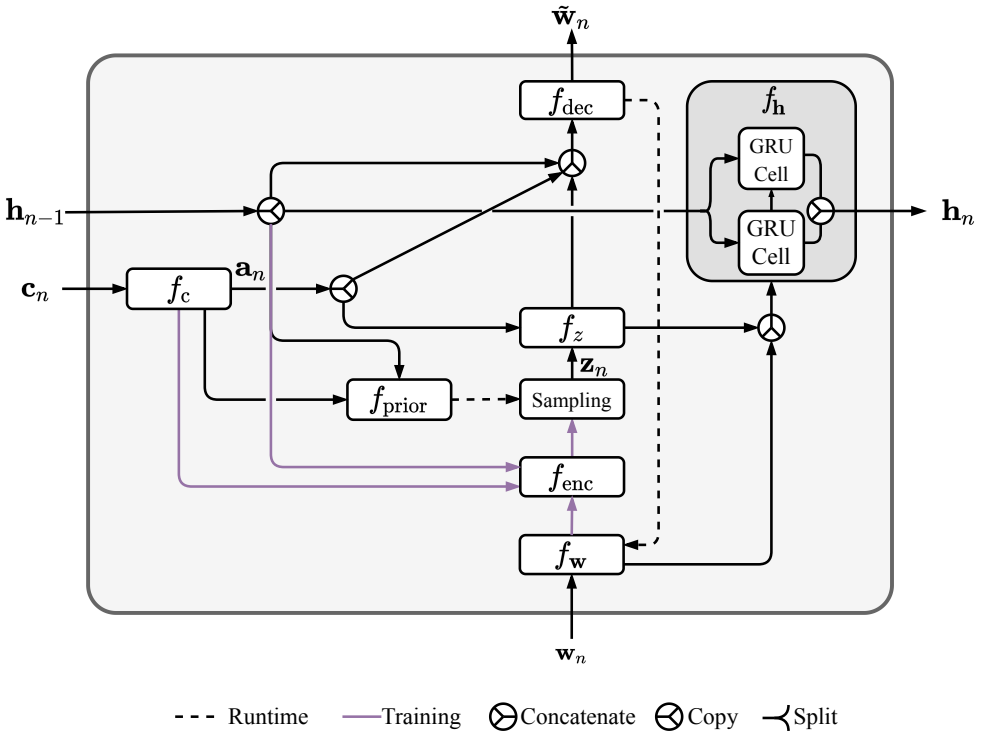


Figure 5.3: Internal structure of Motion Cell. A Motion Cell can be viewed as a recurrent unit conditioned on control signals.

of poses

$$\tilde{\mathbf{X}}_{nl:(n+1)l} = f^{\mathbf{w}}_{\text{dec}}(\tilde{\mathbf{w}}_n), \tag{5.3}$$

where $\tilde{\mathbf{w}}_n$ is the output of the Motion Cell at time-step n (also called the reconstructed Motion Word) and $\tilde{\mathbf{X}}_{nl:(n+1)l}$ is the corresponding reconstructed sub-sequence. The details of the Motion Word encoder ($f^{\mathbf{w}}_{\text{enc}}$) and decoder ($f^{\mathbf{w}}_{\text{dec}}$) are given in Table 5.1, and the next section describes the internal structure of a Motion Cell.

5.3.3 Motion Cell

Our recurrent model is constructed by a probabilistic recurrent unit called a Motion Cell. The design of a Motion Cell is based on an entangled conditional VAE and a transition block. The VAE models the spatial dependencies and is additionally conditioned on control parameters and previous information. The transition block models the temporal dependencies and is a function of not only the input variable and previous internal state but also the current latent variable. By conditioning both spatial and temporal paths on the latent variable, we introduce variability across

Algorithm 1 This algorithm represents the FORWARD process of a Motion Cell for a single time-step during **training**. It takes as input a motion word $\mathbf{w}_n$, control signal $\mathbf{c}_n$, and previous internal state $\mathbf{h}_{n-1}$, and outputs a motion word $\tilde{\mathbf{w}}_n$ and updates the internal state to $\mathbf{h}_n$

function FORWARD($\mathbf{w}_n$, $\mathbf{h}_{n-1}$, $\mathbf{c}_n$)
  `/* Compute attribute vector */`
  $\mathbf{a}_n = f_{\mathrm{c}}(\mathbf{c}_n)$
  `/* Compute posterior distribution */`
  $\boldsymbol{\mu}_{q,n} = f_{\mathrm{enc}}^{\boldsymbol{\mu}}\left(f_{\mathbf{w}}\left(\mathbf{w}_n\right), \mathbf{h}_{n-1}, \mathbf{a}_n\right)$
  $\boldsymbol{\sigma}_{q,n} = f_{\mathrm{enc}}^{\boldsymbol{\sigma}}\left(f_{\mathbf{w}}\left(\mathbf{w}_n\right), \mathbf{h}_{n-1}, \mathbf{a}_n\right)$
  `/* Sample latent variable from posterior distribution */`
  $\mathbf{z}_n \sim \mathcal{N}(\mathbf{z}_n; \boldsymbol{\mu}_{q,n}, \operatorname{diag} \boldsymbol{\sigma}_{q,n}^2)$
  `/* Compute prior distribution */`
  $\boldsymbol{\mu}_{p,n} = f_{\mathrm{prior}}^{\mu}\ (\mathbf{h}_{n-1}, \mathbf{a}_n)$
  $\boldsymbol{\sigma}_{p,n} = f_{\mathrm{prior}}^{\sigma}\ (\mathbf{h}_{n-1}, \mathbf{a}_n)$
  `/* Update internal state */`
  $\mathbf{h}_n = f_{\mathbf{h}}\left(f_{\mathbf{w}}\left(\mathbf{w}_n\right), f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}\right)$
  `/* Compute cell output */`
  $\tilde{\mathbf{w}}_n = f_{\mathrm{dec}}\left(f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}, \mathbf{a}_n\right)$
  return ($\tilde{\mathbf{w}}_n$, $\mathbf{h}_n$, $\boldsymbol{\mu}_{q,n}$, $\boldsymbol{\sigma}_{q,n}$, $\boldsymbol{\mu}_{p,n}$, $\boldsymbol{\sigma}_{p,n}$)
end function

both dimensions. The structure of a Motion Cell is illustrated in Figure 5.3. In the following, we describe in more detail how Motion Cells operate during the training and generation phases.

5.3.3.1 Training Phase

Algorithm 1 provides the processing steps of a Motion Cell for a single time-step during training (the FORWARD function). Unlike a standard VAE, the posterior is not only conditioned on the input but also on the previous internal state and control parameters. A computationally inexpensive and common choice for the latent code distribution is a factorized Gaussian distribution

$$\begin{aligned} q(\mathbf{z}_n|\mathbf{w}_n, \mathbf{h}_{n-1}, \mathbf{a}_n) &= q(\mathbf{z}_n|\mathbf{w}_{\leq n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n}) \\ &= \mathcal{N}(\mathbf{z}_n; \boldsymbol{\mu}_{q,n}, \operatorname{diag}(\boldsymbol{\sigma}_{q,n}^2)), \end{aligned} \tag{5.4}$$

where $\mathbf{z}_n \in \mathbb{R}^{D_z}$ is the latent variable, $\mathbf{h}_n \in \mathbb{R}^{D_h}$ is the internal state of the Motion Cell which summarizes all the past information up to step n, and $\mathbf{a}_n = f_{\mathrm{c}}(\mathbf{c}_n) \in \mathbb{R}^{D_a}$ is the attribute vector extracted from control signals. In our model we only used weak attributes such as action type or style, either included as a component of sample labelling or inferred by integrated classifiers if the sample is unlabelled. However, the same methods can be straightforwardly extended to include

other attributes, including dynamic parameters of the motion such as locomotion trajectory. We set $D_z = 96$, $D_h = 1024$, and $D_a = 8$.

The mean, $\boldsymbol{\mu}_{q,n}$, and covariance parameters, $diag(\boldsymbol{\sigma}_{q,n})$, are computed as:

$$\begin{aligned} \boldsymbol{\mu}_{q,n} &= f_{\text{enc}}^{\boldsymbol{\mu}} \left(f_{\mathbf{w}} \left(\mathbf{w}_n \right), \mathbf{h}_{n-1}, \mathbf{a}_n \right), \\ \boldsymbol{\sigma}_{q,n} &= f_{\text{enc}}^{\boldsymbol{\sigma}} \left(f_{\mathbf{w}} \left(\mathbf{w}_n \right), \mathbf{h}_{n-1}, \mathbf{a}_n \right), \end{aligned} \tag{5.5}$$

where $f_{\text{enc}}^{\boldsymbol{\mu}}$ and $f_{\text{enc}}^{\boldsymbol{\sigma}}$ are non-linear complex functions such as multilayer perceptrons (MLP). $f_{\mathbf{w}}$ is also implemented as an MLP for extracting Motion Word features, which is an essential requirement for learning complex motions. During training the latent variable is sampled from the posterior distribution using the reparameterization trick [11]:

$$\mathbf{z}_n \sim \mathcal{N}\left(\mathbf{z}_n; \mu_{q,n}, \operatorname{diag} \sigma_{q,n}^2 \right) \tag{5.6}$$

Similar to the posterior distribution, the prior distribution is also conditioned on the previous internal state and attribute vectors:

$$\begin{aligned} p(\mathbf{z}_n | \mathbf{h}_{n-1}, \mathbf{a}_n) &= p(\mathbf{z}_n | \mathbf{w}_{<n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n}) \\ &= \mathcal{N}(\mathbf{z}_n; \boldsymbol{\mu}_{p,n}, \operatorname{diag} \boldsymbol{\sigma}_{p,n}^2), \end{aligned} \tag{5.7}$$

where:

$$\begin{aligned} \boldsymbol{\mu}_{p,n} &= f_{\text{prior}}^{\boldsymbol{\mu}} \left(\mathbf{h}_{n-1}, \mathbf{a}_n \right), \\ \boldsymbol{\sigma}_{p,n} &= f_{\text{prior}}^{\boldsymbol{\sigma}} \ \left(\mathbf{h}_{n-1}, \mathbf{a}_n \right). \end{aligned} \tag{5.8}$$

where $f_{\text{prior}}^{\boldsymbol{\mu}}$ and $f_{\text{prior}}^{\boldsymbol{\sigma}}$ are implemented as MLPs. Conditioning the prior and posterior distributions on past information increases the temporal representational power of the model. Additionally, conditioning them on control parameters helps the model find distinct modes within the latent space.

In contrast to standard RNNs in which the output distribution is only conditioned on the previous internal state, the output distribution in the Motion Cell is also conditioned on the latent

variable and control signals:

$$p(\mathbf{w}_n|\mathbf{z}_n, \mathbf{h}_{n-1}, \mathbf{a}_n) = p(\mathbf{w}_n|\mathbf{w}_{<n}, \mathbf{z}_{\leq n}, \mathbf{a}_{\leq n}). \tag{5.9}$$

In this work, we formulate the VAE decoder function deterministically, such that the reconstructed output, $\tilde{\mathbf{w}}_n$, is computed by an MLP:

$$\tilde{\mathbf{w}}_n = f_{\text{dec}}\left(f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}, \mathbf{a}_n\right), \tag{5.10}$$

where $f_{\mathbf{z}}$ is a feature extraction MLP applied on the latent variable.

The internal state of the Motion Cell is updated by a transition function given the current input, previous internal state, and current latent variable:

$$\mathbf{h}_n = f_{\mathbf{h}}\left(f_{\mathbf{w}}\left(\mathbf{w}_n\right), f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}\right). \tag{5.11}$$

Conditioning the internal state on the latent variable $\mathbf{z}_n$ makes the temporal transition probabilistic and also helps the model address the mean collapse problem. Similar to [29], we used two stacked Gated Recurrent Units (GRU) with an internal state of size 512 for the transition function where the Motion Cell internal state is formed by concatenating the internal state of the two GRU cells. All of the components of the Motion Cell are learned by optimizing the objective function explained in Section 5.3.5.

5.3.3.2 Generation Phase

Algorithm 2 provides the processing steps for a single time-step of a Motion Cell during motion synthesis (the SAMPLE function). At each time step during generation the latent variable is sampled from a prior distribution, computed in the same manner as the posterior distribution sampling done in the training phase (Eq. 5.7):

$$\mathbf{z}_n \sim \mathcal{N}\left(\mathbf{z}_n; \boldsymbol{\mu}_{p,n}, \operatorname{diag}\sigma^2_{p,n}\right). \tag{5.12}$$

Algorithm 2 This algorithm represents the SAMPLE process of a Motion Cell for a single time-step during **generation**. It takes control signal $\mathbf{c}_n$ and previous internal state $\mathbf{h}_{n-1}$, and generates motion word $\tilde{\mathbf{w}}_n$ and current internal state $\mathbf{h}_n$

function SAMPLE($\mathbf{h}_{n-1}$, $\mathbf{c}_n$)
  `/* Compute attribute vector */`
  $\mathbf{a}_n = f_{\mathrm{c}}(\mathbf{c}_n)$
  `/* Compute prior distribution parameters */`
  $\boldsymbol{\mu}_{p,n} = f^{\mu}_{\mathrm{prior}}\ (\mathbf{h}_{n-1}, \mathbf{a}_n)$
  $\boldsymbol{\sigma}_{p,n} = f^{\sigma}_{\mathrm{prior}}\ (\mathbf{h}_{n-1}, \mathbf{a}_n)$
  `/* Sample latent variable from prior distribution */`
  $\mathbf{z}_n \sim \mathcal{N}(\mathbf{z}_n; \boldsymbol{\mu}_{p,n}, \operatorname{diag} \boldsymbol{\sigma}^2_{p,n})$
  `/* Compute cell output */`
  $\tilde{\mathbf{w}}_n = f_{\mathrm{dec}}\left(f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}, \mathbf{a}_n\right)$
  `/* Update internal state */`
  $\mathbf{h}_n = f_{\mathbf{h}}\left(f_{\mathbf{w}}\left(\tilde{\mathbf{w}}_n\right), f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}\right)$
  return ($\tilde{\mathbf{w}}_n$, $\mathbf{h}_n$)
end function

The latent variable is then used with the previous internal state and control signals to generate the reconstructed Motion Word $\tilde{\mathbf{w}}_n$ (Eq. 5.10). Finally, the internal state is updated using the previous internal state, current latent vector, and the reconstructed Motion Word:

$$\mathbf{h}_n = f_{\mathbf{h}}\left(f_{\mathbf{w}}\left(\tilde{\mathbf{w}}_n\right), f_{\mathbf{z}}\left(\mathbf{z}_n\right), \mathbf{h}_{n-1}\right). \tag{5.13}$$

5.3.4 Attributes Classifiers

For each attribute type, we integrate a separate pre-trained classifier into the generative model. Integrating classifiers into the hierarchical probabilistic recurrent network serves three purposes. First, they provide control parameters to the generative model for unlabelled data, allowing our system to operate in a semi-supervised manner (dashed arrows in Figure 5.2). Second, during training, the classifiers provide additional high-level signals (both from their intermediate layers as well as the output class inferred by the classifier) to the objective function. This constrains the generative model to generate motions semantically coherent with the motion attributes (see Section 5.3.5.2). Third, the classifiers can be used for the evaluation of our generative model (see Section 5.5).

We implemented all the classifiers using one-dimensional convolutional neural networks and trained them on 50% of the training data. We observed that this amount of training data is sufficient

to label the rest of the data with a high accuracy. Further details of classifier implementation is given in Section 5.4.

5.3.5 Objective Function

We formulate model training as an optimization problem by minimizing the objective function

$$\mathcal{L} = \mathcal{L}_{RVAE} + \lambda_{CL}\mathcal{L}_{CL} + \lambda_{Ang}\mathcal{L}_{Ang}, \tag{5.14}$$

where $\mathcal{L}_{RVAE}$ is the recurrent VAE loss equal to the sum of the negative step-wise variational lower bound over the whole sequence. We define a new hierarchical geodesic loss for the reconstruction part of $\mathcal{L}_{RVAE}$ which is more accurate than the l_p norm loss and takes into account the relative impact of each joint in the kinematic tree on the final loss. $\mathcal{L}_{CL}$ is the complementary loss provided by the classifiers, computed by evaluating the ground-truth and reconstructed samples in the intermediate and last layer of each classifier. $\mathcal{L}_{Ang}$ is the sum of constraints encouraging the model to produce valid joint representation. We will describe each term in more detail below.

5.3.5.1 RVAE Objective

The first term in our objective function, $\mathcal{L}_{RVAE}$, is defined as a variational autoencoder objective summed over all sequence steps as follows:

$$\begin{aligned}\mathcal{L}_{RVAE} = \mathbf{E}_{q(\mathbf{z}_{\leq N}|\mathbf{w}_{\leq N},\mathbf{a}_{\leq N})}\Bigg[&\sum_{n=1}^{N=T/l} -\log p\left(\mathbf{w}_n|\mathbf{z}_{\leq n}, \mathbf{w}_{<n}, \mathbf{a}_{\leq n}\right) \\ &+ \lambda_{KL}\mathrm{KL}(q(\mathbf{z}_n|\mathbf{w}_{\leq n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n})\|p(\mathbf{z}_n|\mathbf{w}_{<n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n}))\Bigg]. \\ &= \mathcal{L}_{rec} + \lambda_{KL}\mathcal{L}_{KL}.\end{aligned} \tag{5.15}$$

The first term in the above loss is the expected log-likelihood or reconstruction loss, usually defined as the distance between observations and the reconstructed values. We define our reconstruction term as a custom loss over joint angles rather than Motion Words to simultaneously train the Motion Word encoder $f^{\mathbf{w}}_{\text{enc}}$ and decoder $f^{\mathbf{w}}_{\text{dec}}$. The second term in the summation is the KL-divergence between the posterior and the prior at time-step n weighted by λ_{KL}. To prevent optimization process from getting stuck in an undesirable stable equilibrium we used an annealing scheduler for

λ_{KL} where the optimization is performed for a few epochs with $\lambda_{KL} = 0$ (warm-up phase), then λ_{KL} is slowly increased from 0 to 1 (annealing phase), and then for the last few epochs we set $\lambda_{KL} = 1$ (cool-down phase) [166].

In the following we explain how the RVAE objective function in Eq. 5.15 is derived. We can consider the whole sequence as a single sample and write the ELBO similar to Eq. 2.17 as follows:

$$\mathcal{E}_{RVAE} = \mathbf{E}_{q_\phi(\mathbf{z}_{\leq N}|\mathbf{w}_{\leq N},\mathbf{a}_{\leq N})} \log \frac{p_\theta(\mathbf{w}_{\leq N}, \mathbf{z}_{\leq N}|\mathbf{a}_{\leq N})}{q_\phi(\mathbf{z}_{\leq N}|\mathbf{w}_{\leq N}.\mathbf{a}_{\leq N})} \tag{5.16}$$

By factorizing p_θ and q_ϕ across time we have:

$$\begin{aligned}
\mathcal{E}_{RVAE} &= \mathbf{E}_{q_\phi(\mathbf{z}_{\leq N}|\mathbf{w}_{\leq N},\mathbf{a}_{\leq N})} \log \frac{\prod_{n=1}^{N} p_\theta(\mathbf{w}_n, \mathbf{z}_n|\mathbf{w}_{<n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n})}{\prod_{n=1}^{N} q_\phi(\mathbf{z}_n|\mathbf{z}_{<n}\mathbf{w}_{<n}.\mathbf{a}_{\leq n})} \\
&= \mathbf{E}_{q_\phi(\mathbf{z}_{\leq N}|\mathbf{w}_{\leq N},\mathbf{a}_{\leq N})} \log \prod_{n=1}^{N} \frac{p_\theta(\mathbf{w}_n|\mathbf{w}_{<n}, \mathbf{z}_{\leq n}, \mathbf{a}_{\leq n}) p_\theta(\mathbf{z}_n|\mathbf{z}_{<n}, \mathbf{a}_{\leq n})}{q_\phi(\mathbf{z}_n|\mathbf{z}_{<n}\mathbf{w}_{<n}.\mathbf{a}_{\leq n})} \\
&= \mathbf{E}_{q_\phi(\mathbf{z}_{\leq N}|\mathbf{w}_{\leq N},\mathbf{a}_{\leq N})} \Bigg[\sum_{n=1}^{N} \log p_\theta\left(\mathbf{w}_n|\mathbf{w}_{<n}, \mathbf{z}_{\leq n}, \mathbf{a}_{\leq n}\right) \\
&- \mathrm{KL}(q_\phi(\mathbf{z}_n|\mathbf{w}_{\leq n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n}) \| p_\theta(\mathbf{z}_n|\mathbf{w}_{<n}, \mathbf{z}_{<n}, \mathbf{a}_{\leq n})) \Bigg],
\end{aligned} \tag{5.17}$$

where the RVAE objective is the negative value of the above ELBO ($\mathcal{L}_{RVAE} = -\mathcal{E}_{RVAE}$). In the following we describe how the reconstruction loss is formulated.

Geodesic Distance of Joint Angles: Assuming a deterministic prediction in joint angles, the first term can be defined as a reconstruction loss. Often, metrics in the Euclidean space such as l_1 and l_2 norms are used as the reconstruction loss. However, these metrics do not represent the geodesic distance of two rotations which confuses the training process, especially for large angular distances and at the beginning of the optimization process. To address the above-mentioned problems in Euclidean distances, we define more relevant distance functions which respect the intrinsic structure of 3D rotations both for quaternions and rotation matrices. The angular distance between two unit quaternions $\mathbf{q}$ and $\tilde{\mathbf{q}}$ is defined as:

$$d(\mathbf{q}, \tilde{\mathbf{q}}) = \mathbf{q}\tilde{\mathbf{q}}^{-1} = 2 \arccos\left(\mathbf{q} \cdot \tilde{\mathbf{q}}\right). \tag{5.18}$$

Since the quaternions double-cover the space of rotations meaning that quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation we can take into account this ambiguity by modifying the above function as:

$$d_1(\mathbf{q}, \tilde{\mathbf{q}}) = 2 \arccos\left(|\mathbf{q} \cdot \tilde{\mathbf{q}}|\right) \tag{5.19}$$

Since arccos is a monotonically decreasing function we can define an approximate but computationally less expensive distance as:

$$d_2(\mathbf{q}, \tilde{\mathbf{q}}) = 1 - |\mathbf{q} \cdot \tilde{\mathbf{q}}|. \tag{5.20}$$

which only needs 4 multiplication and 1 comparison for each pair of quaternions [167]. It can be proved that the square of the l_2 norm of two unit quaternions is equivalent to Eq.5.20 for small angular distances:

$$\begin{aligned} \|\mathbf{q} - \tilde{\mathbf{q}}\|^2 &= \|\mathbf{q}\|^2 + \|\tilde{\mathbf{q}}\|^2 - 2(\mathbf{q} \cdot \tilde{\mathbf{q}}) \\ &= 2(1 - \mathbf{q} \cdot \tilde{\mathbf{q}}). \end{aligned} \tag{5.21}$$

Similarly, we can modify the above measure to disambiguate the quaternions representations as follows:

$$d_3(\mathbf{q}, \tilde{\mathbf{q}}) = \min\left\{\|\mathbf{q} - \tilde{\mathbf{q}}\|^2, \|\mathbf{q} + \tilde{\mathbf{q}}\|^2\right\} \tag{5.22}$$

All distance measures d_1, d_2, and d_3 address the double-coverage problem, however, the last two are approximations and do not measure the exact geodesic distance.

Similarly, for the scenarios where the joint angles are represented by rotation matrices, we can use the Geodesic distance between a pair of rotation matrices similar to [168] and defined as:

$$d(\mathbf{R}, \tilde{\mathbf{R}}) = \|\log(\tilde{\mathbf{R}}\mathbf{R}^\top)\|, \tag{5.23}$$

where $\tilde{\mathbf{R}}\mathbf{R}^\top$ is the difference rotation matrix, $\log(\tilde{\mathbf{R}}\mathbf{R}^\top)$ is a skew-symmetric matrix containing the rotation axis-angle components and $\|\log(\tilde{\mathbf{R}}\mathbf{R}^\top)\|$ is the magnitude of the angular distance multiplied by a constant.

Hierarchical Loss: Proposed approaches in human motion modelling represent each human pose either by 3D joint locations in a global or body's local coordinate system or 3D joint angles where,

given the limbs' length, the final position and orientation of the body parts are calculated by forward kinematics. Models which use 3D joint locations usually normalize the skeleton size of the training samples and define the loss as an l_p norm over joint locations [3, 4]. The main problem in such approaches is that during training and generation, they are not exploiting the constraints imposed by parameterized skeleton and limbs rigidity. Therefore, the generation phase should be followed by a corrective re-projection onto a valid character skeleton.

Modelling poses by joint angles inherently follows the constraints imposed by parameterized skeleton [169, 2, 1, 67]. However, defining loss over joint angles ignore the amount of influence that each joint contributes to the learning process and gives all joints equal weights. On the other hand, an error in a parent joint has more impact on the final pose than the same amount of error in its child joints. This is due to the fact that an error in the parent joints propagates through all of its children down to the leaf nodes in the kinematic tree during forward kinematics. Recently, [29] proposed using joint angles to represent body pose but defined the loss over joint locations by applying a differentiable forward kinematics on ground-truth and predicted joint angles. However, applying forward kinematics at each pose is computationally expensive, especially for long sequences and when the number of joints is high.

In this work, we propose a hierarchical loss over joint angles which weights each joint's error based on its impact on the reconstructed pose as follows:

$$\mathcal{L}_{rec}(t) = \sum_{k=1}^{K} \alpha_k d(\mathbf{X}_t^k, \tilde{\mathbf{X}}_t^k), \tag{5.24}$$

where $\mathbf{X}_t^k$ and $\tilde{\mathbf{X}}_t^k$ are the ground-truth and the reconstructed joint angles for joint k at time t and $d(.)$ is one of the distance functions defined in Eq.5.19, 5.20, 5.22, or 5.23. α_k is the impact factor which weights the impact of the corresponding joint angle on the pose reconstruction. A rule of thumb for choosing α_ks is that the child joint should be weighted with a lower impact factor compared to its parent joint ($\alpha_k < \alpha_{parent(k)}$) in the kinematic tree. In this work, we set α_k as the maximum path length from joint k down to all of the connected end-effectors in an average body skeleton. We can define α_k recursively as follows:

$$\alpha_k = \max_j(\alpha_j + l_{k-j}), j \in SC_k, \tag{5.25}$$

where SC_k is the set of all children of joint k, and l_{k-j} is the length of the bone connecting joints l and j. As suggested by [29] we also evaluated the results by applying forward kinematics and computing the positional loss. In practice, the results were very close, while the latter took around 35% longer for training.

5.3.5.2 Classifiers Loss

The classifiers are trained to infer the motion attributes and incorporate a complementary loss from the output of intermediate and final layers. This complementary loss can be defined as:

$$\mathcal{L}_{CL}(t) = \sum_{c=1}^{C} \sum_{l \in L_c} \beta_{(c,l)} d(l(\mathbf{X}_t), l(\tilde{\mathbf{X}}_t)), \tag{5.26}$$

where C is the number of classifiers and L_c is the set of layers in cth classifier. $d(l(\mathbf{X}), l(\tilde{\mathbf{X}}))$ computes the loss for the output of layer l given ground-truth and reconstructed samples. $\beta_{(c,l)}$ is a predefined weight assigned for each layer. We compute the loss by using the l_2 norm for intermediate layers and cross-entropy loss for the attribute labels. Details of the classifiers' architecture are given in Table 5.1.

5.3.5.3 Intrinsic Rotation Representation Constraints

In order to encourage the model to produce valid rotations, we add some constraint terms to the final objective function based on the representation we use for the joint angles. This helps better ensure convergence at the beginning of the training process and smooths the optimization landscape. Although we normalize the output of $f_{\text{dec}}^{\mathbf{w}}$, better performance is achieved when these outputs are very close to valid rotations leaving the role of normalizers as only a final correction on very small errors.

For rotation matrices, we define two constraints: orthogonality and unit determinant, formulated as follows:

$$\begin{aligned} \mathcal{L}_{\text{ang}}(t) &= c_1 \mathcal{L}_{\text{orth}}(t) + c_2 \mathcal{L}_{\text{det}}(t) \\ &= \sum_{k=1}^{K} \left(c_1 \|\tilde{R}_t^k (\tilde{R}_t^k)^\top - I\|_2^2 + c_2 |\det(\tilde{R}_t^k) - 1| \right) \end{aligned} \tag{5.27}$$

where the first term encourages the orthogonality of the output matrices and the second term

enforces the model to produce matrices with a unit determinant. We also added Sigmoid activation to the output of $f_{\text{dec}}^{\mathbf{w}}$ to ensure that the elements of the output matrices are in the range of $[0, 1]$.

For quaternions we only need to set the unit length constraint

$$\mathcal{L}_{\text{ang}} = \mathcal{L}_{q-norm} = \sum_{k=1}^{K} |\|\tilde{\mathbf{q}}_t^k\|_2^2 - 1| \tag{5.28}$$

For axis-angle rotation representation, we did not set any constraint as they represent the three degrees of freedom by only three scalars.

5.4 Implementation and Training

5.4.1 Dataset

We trained and evaluated our model on a subset of AMASS [15], a very large database of human motion that unifies different marker-based motion capture datasets by representing them in a common framework. The kinematic tree is represented by 21 joints and the root (pelvis). We used the MoVi [161] and RuB [57] datasets from AMASS for training and evaluating the main module and the rest of the AMASS data for training the denoising autoencoder.

The control parameters in our model are action type and gender. We used a subset of actions: walking, jogging, jumping, and lifting. The data were split into 150, 25, and 25 subjects for the purpose of training, validation, and testing, respectively. All splits contained male and female subjects in equal proportion.

5.4.2 Training Process

The details of the model architecture are given in Table 5.1. All model components were implemented using the PyTorch library.

For training the hierarchical model (Motion Cell, $f_{\text{enc}}^{\mathbf{w}}$, and $f_{\text{dec}}^{\mathbf{w}}$), we optimized the objective function in Eq.5.14 using the Adam optimizer [170] with a learning rate of 0.001, no weight decay, and a batch size of 64. We also set the gradient norm clipping to 0.1 to avoid any exploding gradients. All weights of the model were initialized using Kaiming initialization [171]. Depending on the setting and control parameters, the number of training epochs ranges between 120 and 160.

Function	Architecture
f_w, f_z	$2 \times$ [FC(128) + ELU] + FC(96) + ELU
f_{dec}	$2 \times$ [FC(128) + ELU] + FC(96) + ELU
f^{μ}_{enc}, f^{μ}_{prior}	$4 \times$ [FC(128) + ELU] + FC(96)
f^{σ}_{enc}, f^{σ}_{prior}	$4 \times$ [FC(128) + ELU] + FC(96) + Softplus
f_h	$2 \times$ GRUCell(512)
$f^{\mathbf{w}}_{\text{enc}}$	$2 \times$ [FC(128) + ELU] + FC(96) + ELU
$f^{\mathbf{w}}_{\text{dec}}$	$2 \times$ [FC(128) + ELU] + FC($3 \times D_P$)
Classifiers	$3 \times$ [Conv1D + ReLU] + AdaptiveAvgPool1D + FC(N_C)
Denoising Autoencoder	$2 \times$ [Conv1D + ReLU] + ConvTranspose1D + ReLU + ConvTranspose1D

Table 5.1: The architecture of model components. FC(n) is the abbreviation for Fully Connected linear layer with n nodes. Conv1D and ConvTranspose1D are one-dimensional convolution and transposed convolution layers. AdaptiveAvgPool1D is one-dimensional adaptive average pooling layer.

Figure 5.4 shows how we set the schedulers for loss coefficients in Eq. 5.14. We set an annealing scheduler for λ_{KL} to address the KL vanishing problem. We also set λ_{CL} to zero for the beginning of the training and then gradually increase it. These two strategies allow the model to focus more on capturing useful information for reconstruction during the initial epochs.

For each combination of attributes, the initial internal states of the GRU cells are learned as a Gaussian distribution. Then each sequence is initialized by sampling the initial state from the distribution, which corresponds to the required attribute.

We train our recurrent model in a teacher forcing scheme (i.e. the ground-truth input is provided to the Motion Cell at each time-step during training). Although this is an effective and fast approach for training, the model is prone to exposure bias and risks overfitting the training data. To address this problem, we experimented with three different mitigating strategies: (i) progressively corrupting input by adding Gaussian noise [1, 67], (ii) progressively dropping motion words and exposing the model to its own previous output [2, 29], and (iii) adding joint-wise dropout on the input poses [172]. We achieved the best results when we used the second strategy with a scheduled drop rate.

We trained all classifiers with similar architecture (Table 5.1) on 50% of training data using the

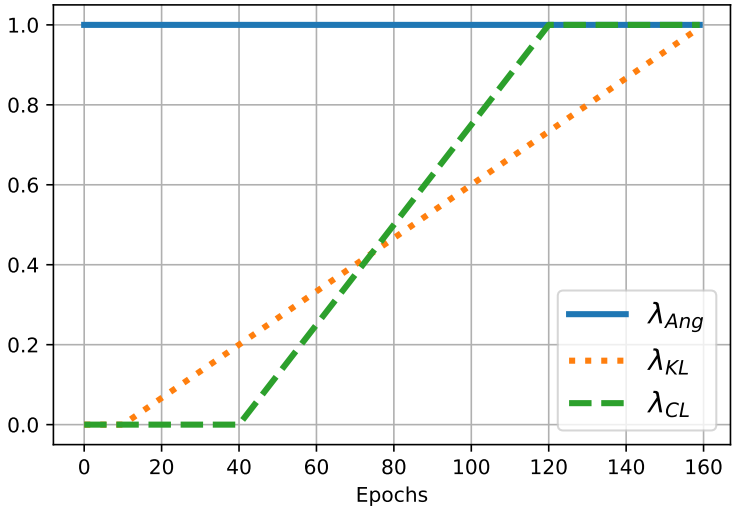


Figure 5.4: Scheduling the coefficients for each term in Eq. 5.14 during training.

Adam optimizer with a learning rate of 0.005 for 30 epochs. We used Adaptive Average Pooling before the last fully connected layer to adapt the classifier models to different input lengths.

We trained the denoising autoencoder separately and on the rest of the AMASS data. During training 10% of the input, dimensions were chosen randomly and corrupted by Gaussian noise with zero mean and standard deviation of 0.5. We trained this model for 300 epochs and used the Adam optimizer with a learning rate of $1e-4$ with exponential decay of 0.99 per 10 epoch.

5.5 Experiments and Evaluation

5.5.1 Sequence Samples

We present representative samples of the output of our model in Figure 5.5. You can see by comparing the generated samples (orange) to real samples (blue), that our synthesized samples look very natural and from the same action cluster. Please also refer to our video demo for additional visual results [1].

[1] https://www.youtube.com/watch?v=r9F74LcGC0A&t=194s

Figure 5.5: Samples of real (blue) and synthetic (orange) motion sequences.

5.5.2 Models and Ablations

For the purpose of comparison, we compared our model with Pavllo et al.'s Quaternet [29] and Fragkiadaki et al.'s Encoder-Recurrent-Decoder (ERD) model [1]. These two models were trained on the same training data with the same training hyper-parameter optimization techniques as our model. The initial internal state was learned in the same way to our model. Sampling the initial state is the only source of stochasticity in these two models.

For all models, we used a common generation scheme. Each walking or jogging sequence was generated with 140 frames, and the first 20 frames were discarded during evaluation (resulting in 4 seconds of motion). For the non-periodic actions (jumping and lifting), we terminated the generated sequence when they collapsed to the mean pose.

In order to evaluate the influence of model components, we trained three additional ablated configuration of the our model. In the first ablated configuration, "Proposed(SL)" (for "Single Layer"), we removed the hierarchical encoder $f_{\text{enc}}^{\mathbf{w}}$ and decoder $f_{\text{dec}}^{\mathbf{w}}$, and fed the individual poses directly to the recurrent model. For the second ablation configuration, "Proposed(NL)" (for "Normal Loss"), we disabled the influence of hierarchical loss by setting all α_k coefficients in the Eq. 5.24 to 1. The last ablation, "Proposed(NC)" (for "No Classifier"), disabled the influence of classifiers on the final loss by setting λ_{CL} to zero.

For all evaluated models, we achieved comparable quantitative results between quaternion and rotation matrices representations, both of which outperformed axis-angle representations. Therefore, for the rest of the chapter, we only report the results for quaternions.

5.5.3 Quantitative Evaluation

In this work, we evaluate models based on two main criteria: quality and diversity. We expect the generated samples to be realistic and coherent with the attributes which are set as control parameters (quality). In addition, we expect the model to generate motions with high diversity and natural stochasticity while still following the manifold of realistic motions (diversity). To codify both criteria in our quantitative evaluation we use the *Inception Score (IS)* [163] and *Fréchet Inception Distance Score (FID)* [164] metrics which were originally proposed for image generative models. Both evaluation metrics have been shown to correlate well with human evaluation of

generated images.

IS is formulated based on two criteria, diversity and quality, defined as follows:

$$\text{IS} = \exp\left(\mathbf{E}_{\tilde{\boldsymbol{X}}\sim p_g} D_{KL}(p(\mathbf{a}|\tilde{\mathbf{X}})\|p(\mathbf{a}))\right) \tag{5.29}$$

where $\tilde{\mathbf{X}}$ is a synthetic sample generated by a generative model, $p(\mathbf{a}|\tilde{\mathbf{X}})$ is the conditional attribute distribution of a classifier which is pre-trained on separate training data, and $p(\mathbf{a}) = \int_{\tilde{\mathbf{X}}} p(\mathbf{a}|\tilde{\mathbf{X}}) p_{\text{model}}(\tilde{\mathbf{X}})$ is the marginal attribute distribution. Equation 5.29 can be also written as:

$$\text{IS} = \exp\left(H(\mathbf{a}) - H(\mathbf{a}|\tilde{\mathbf{X}})\right), \tag{5.30}$$

where $H(\mathbf{a})$ and $H(\mathbf{a}|\tilde{\mathbf{X}})$ are the attribute entropy and the conditional attribute entropy, respectively. Generated animations which fulfil the semantics defined by the attributes should have a conditional attribute distribution $p(\mathbf{a}|\tilde{\mathbf{X}})$ with low entropy. In other words, the classifier should be very confident about the attribute associated with the generated animation. On the other hand, we expect our model to generate a high variety of motions for each attribute class, therefore, $p(\mathbf{a})$ should have a high entropy. An estimator of IS as follows:

$$\text{IS} \approx \exp\left(\frac{1}{M}\sum_{i=1}^{M} D_{KL}\left(p(\mathbf{a}|\tilde{\mathbf{X}}^{(i)})\|\hat{p}(\mathbf{a})\right)\right), \tag{5.31}$$

where $\tilde{\mathbf{X}}^{(i)}$ is a generated motion sample and $\hat{p}(\mathbf{a}) = \frac{1}{M}\sum_{i=1}^{M} p\left(\mathbf{a}|\tilde{\mathbf{X}}^{(i)}\right)$ is the empirical conditional distribution.

FID captures the similarity between the generated and the real motion samples. It evaluates the model by comparing the statistics of a set of generated samples to a set of real motion sequences from the dataset. Similar to IS, we use a classifier trained on a separate dataset. Then, the activations of the last feature extraction layer (the last layer prior to the last fully connected layer) are summarized as a multivariate Gaussian distribution for synthetic and real data. The distance between the two distributions is then computed with Fréchet Distance as follows:

$$\text{FID} = \|\mu_g - \mu_d\|_2^2 + \text{Tr}\left(\Sigma_g + \Sigma_d - 2\left(\Sigma_g\Sigma_d\right)^{\frac{1}{2}}\right), \tag{5.32}$$

Model	IS ↑	FID ↓
Quaternet [29]	5.12	92.31
ERD [1]	5.91	86.42
Proposed(SL)	6.45	31.41
Proposed(NL)	7.11	17.3
Proposed(NC)	7.43	11.92
Proposed	7.52	10.45
Real data	7.64	0

Table 5.2: Results from quantitative evaluations using IS (higher score is better) and FID (lower score is better).

where $\mathcal{N}(\mu_g, \Sigma_g)$ and $\mathcal{N}(\mu_d, \Sigma_d)$ are the distributions of the activations in the last feature extraction layer for synthetic and real data, respectively.

Results: The results of quantitative evaluations are shown in Table 5.2. Using each model, we generated 1000 samples for each combination of attributes ($M = 8000$ in total). Quaternet and ERD generated convincing walking and jogging samples. However, since the only source of stochasticity is the initial hidden state, these models fail to generate a diverse set of sequences (ERD's performance was slightly better due to its hierarchical structure yielding higher diversity at the beginning of the motions). In addition, they usually failed to generate a complete sequence for non-periodic motion such as lifting and were regressed to the mean pose after $70 - 80$ frames. Among ablation configurations, Proposed(SL) had the lowest performance showing the significant influence of hierarchical structure in generating diverse motions (higher $H(\mathbf{a})$). The lower scores for the other two ablation configurations (Proposed(NL) and Proposed(NC)) indicate the impact of our hierarchical loss structure and the effect of integrating classifiers into the loss, respectively, on the conditional attribute entropy and higher motion quality.

5.5.4 Qualitative Evaluation

To qualitatively evaluate how realistic and natural the synthetic animations are, we also performed subjective evaluation experiments. All six evaluated models in the previous section were also used in this experiment. We sampled five sequences for each of the four action types from each model's synthesized sequences, resulting in $6*4*5 = 120$ synthesized samples, which were added to 20 motion

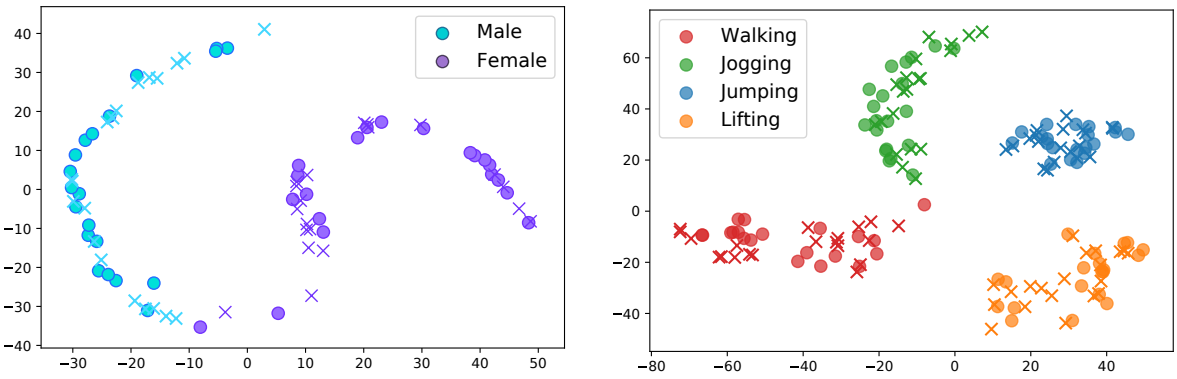
Model	Actions				Average
	Walking	Jogging	Jumping	Lifting	
Quaternet [29]	9.31±0.25	9.45±0.22	6.27±0.31	5.87±0.28	7.73±0.26
ERD [1]	8.97±0.33	8.32±0.30	6.31±0.28	5.71±0.23	7.31±0.28
Proposed(SL)	9.15±0.24	9.26±0.22	8.43±0.37	8.62±0.18	8.87±0.25
Proposed(NL)	9.03±0.29	8.97±0.21	8.25±0.19	8.30±0.32	8.64±0.26
Proposed(NC)	9.32±0.27	9.42±0.30	8.95±0.31	8.92±0.34	9.15±0.27
Proposed	9.38±0.13	9.35±0.12	9.13±0.28	9.27±0.27	9.28±0.21
Real data	9.64±0.13	9.81±0.24	9.62±0.1	9.53±0.21	9.65±0.17

Table 5.3: Results for qualitative evaluation. The results show the average scores assigned to a set of motions sampled from each action set and from each model, where 1 corresponds to completely unrealistic and 10 corresponds to completely realistic.

sequences from real data. 20 participants rated each motion sample from 1 (completely unrealistic) to 10 (completely realistic). The motion clips were displayed to the raters in a randomized order. Raters were asked to rate each animation after it was displayed completely, and no information about the aim of the experiment was given to the raters. We used a few motion clips for the purpose of training the raters before each experiment.

Results: The results of our qualitative evaluation are illustrated in Table 5.3. The qualitative results correlate well with the quantitative results (Pearson correlations of 0.86 and -0.95 with IS and FID, respectively). Quaternet and ERD achieved the lowest ratings for non-periodic actions (jumping and lifting) since they usually fail to complete these motions and instead regressed to the mean pose in the last frames. Among ablation configurations, Proposed(NL) achieved the lowest mean rating, which shows the impact of hierarchical loss on having more realistic motions. Although the main goal of our hierarchical structure is to improve the diversity of motions, the lower ratings achieved by Proposed(SL) compared to the main model nevertheless demonstrate the effectiveness of our hierarchical architecture even on short sequences. Similar to the quantitative evaluation, disabling the hierarchical structure of the loss function (Proposed(NL)) resulted in decreased ratings, suggesting the importance of classifiers to learn action modes on the motion manifold better.

Figure 5.6 shows the visualization of motion sequences in a two-dimensional space. We sampled 20 sequence for each action type and gender from real data (circles) and the sequences generated



(a) Samples for different genders

(b) Samples for different actions

Figure 5.6: Visualizing the activations of the last layer of action classifier projected onto two dimensions using t-sne for real data (circles) and data synthesized data (crosses) by our model. As can be seen, synthesized samples strongly coincide with the corresponding clusters formed by real data.

by our model (crosses), extracting the activations from the last layer of classifiers. We then applied *t-sne* [173] dimensionality reduction to project the activations onto two dimensions. As seen, our model generates sequences with similar diversity to real data while still accurately separating the modes for each action type and gender.

5.6 Conclusion

In this work, we propose a generative motion model focusing on preserving the stochastic nature of human motion while generating convincing and natural spatiotemporal motion sequences. The proposed model uses a deep hierarchical recurrent framework that can be tuned via weak control signals such as action type. Each sequence is generated using a probabilistic recurrent structure which models the underlying stochasticity by injecting noise in an abstract level. We also propose a novel hierarchical geodesic loss that incorporates the structural information of the kinematic tree and compares joint angles based on angular distances, yielding a better representation of error and more accurate learning.

5.7 Future Work

The proposed architecture was implemented for four different action types in addition to the gender attribute. Extending the model to include more actions is not straightforward and is prone to

mode collapse. However, different strategies can possibly be exploited to increase the architecture's capacity for more action types or other additional semantics. One possible solution is to increase the network capacity, though we expect this may make training more difficult. Another solution could be to train a separate network for each subset of attributes or actions, which may serve as a feasible solution since our network is relatively small (around 30MB). Finally, providing additional strong control signals such as body contact with the environment could serve to decrease the uncertainty in the motion generation phase and prevent it from collapsing to the mean pose. These control signals could be provided manually by the animator or by a separate network that is trained on the data [29, 4].

In our recurrent model, each cell is represented as a VAE conditioned on the previous internal state. However, VAEs are based on maximizing a log-likelihood lower bound which might give a suboptimal solution for the true log-likelihood. We also made a strong assumption about the posterior distribution by modelling it as a standard isotropic Gaussian, which could increase the error in the posterior approximation. It is possible that normalizing flows [106, 174] could be exploited to improve this aspect of the model. For example, the approximate posterior distribution could be parameterized by a network of normalizing flows, which is applied to the output of the VAE decoder; this has been shown to provide a tighter lower bound for other applications [175]. Alternatively, the VAE module in our Motion Cell could potentially be replaced entirely with a conditional normalizing flow network in which the temporal dependencies are modelled by conditioning the prior on the previous internal state.

Chapter 6

KinFlow: A Probabilistic Approach for Multi-Modal Character Control

The work in this chapter is in the preparation phase to be submitted to Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation

6.1 Preface

Character control is defined as a framework where the movements of a virtual character are generated given some user control. This is a prevalent scenario in video games where the movement of a character should follow the instructions provided by a gamepad, joystick, or other forms of input. The main challenge in designing such a framework is producing realistic motion that has a high quality, is responsive, and matches the inputs given by the user. All of the approaches used in the industry, such as motion matching [176] and motion graphs [50], are based on the direct use of motion primitives provided in a database. Therefore, these approaches are limited by the amount of data and produce similar motions given the same inputs, which causes repetitive patterns.

In this chapter, we present KinFlow, a new probabilistic approach for character control where the generated motion encompasses the natural variations while closely following the inputs given by the user. Using a new generative framework based on normalizing flows, we produce a character controller which is able to generate realistic locomotion controlled by different control signals. As well as generating high-quality motion with natural variations, our model is extremely compact and compatible with the standard gaming environment requirements, such as execution time.

6.2 Introduction

Even with vast volumes of readily available high-quality motion capture data, generating real-time data-driven controllers for virtual characters has proven difficult. This is because gaming platforms have many strict requirements to be satisfied - the amount of manual preprocessing of data should not be very high, the model must be extremely fast to execute at real-time with very low latency (60 fps or more), must require low memory, must be able to learn all the modalities in the dataset, must be compatible with the standard controlling pipelines. Although many approaches have been proposed and much progress has been made in the field of character animation using machine learning approaches, there is still much room to improve upon the state-of-the-art models to be adopted to the gaming platforms.

The problem can be even more challenging when we aim to produce character animation with the natural variations and randomness seen in human motion. Most machine learning-based approaches

proposed for character control are based on the deterministic modelling of data resulting in the repetitive patterns in the generated data [32, 44, 43, 29, 4]. There have been probabilistic approaches proposed to tackle this problem [33, 27, 177, 25, 168, 178, 12]. However, to our knowledge, none of the proposed methods satisfy all of the requirements to that extent to be adopted by the gaming platforms.

Our proposed architecture is designed as a hierarchical recurrent model constructed by a combination of coupling and auto-regressive flow blocks. The proposed flow-based model provides the exact latent-variable inference and log-likelihood evaluation. Like other flow-based models, the evaluation and synthesis are done without any approximation, which provides accurate inference and enables optimization of the exact log-likelihood. The spatiotemporal dependencies in our model are modelled by modulating all blocks of coupling flows and also adding affine autoregressive connections. In this work, the temporal dependencies are modelled by conditioning not only the affine transformation in the coupling flows but also the 1×1 convolution layer and the newly proposed temporal affine layer to the control signals and past history. This significantly improved both quantitative and qualitative results with less number of coupling layers compared to MoGlow [27], the previous flow-based model for character animation. In addition, the input and output states are modelled similarly to common gaming frameworks for controlling the character. In such state characterization, not only the pose but also the character's movement and the predicted trajectory are modelled probabilistically, delivering a better real-world simulation. In our framework, the character pose space is modelled by not only joint positions but also joint rotations and velocities and blending them in the inference, which achieved much sharper results and was free of bone stretching. Finally, we integrated different strategies to speed up the runtime process to be compatible with the standard gaming platform requirements. The proposed architecture in MoGlow contains 16 affine coupling steps to be expressive enough. In a normal machine, each forward step takes more than 40 ms, equivalent to 25 fps and therefore is not fast enough for real-time game rendering. We exploited several strategies to speed up the runtime process. 1) We reduced the character pose space's dimensionality and conditioning vector by applying PCA or an autoencoder, which significantly helped the optimization step and a more stable training process. 2) We used multi-scale architecture similar to RealNVP [179] and Glow [110]. 3) We applied additional implementation strategies such as Model Pruning [180].

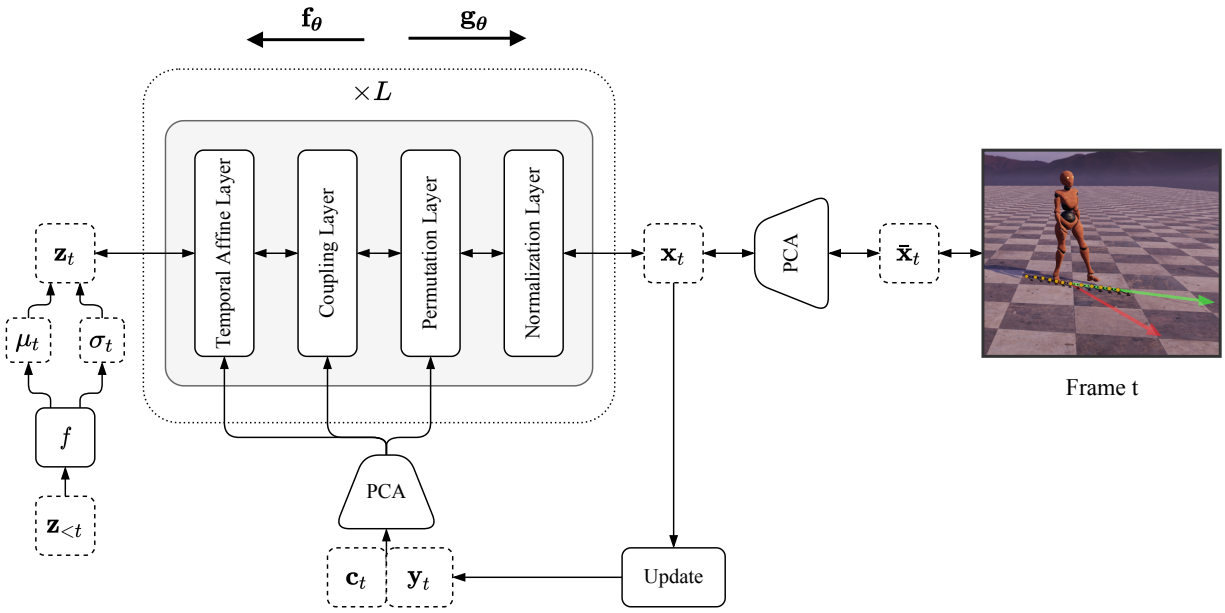


Figure 6.1: An overview of KinFlow. At each step, a sample from base distribution is mapped to the current character state while being conditioned on the control signal and previous states by modulating all invertible transformations.

6.3 System Overview

A visual diagram of KinFlow is shown in Figure 6.1. In the proposed recursive framework, each frame is modelled by the structure of a normalizing flow consisting of a set of invertible and differentiable transformations conditioned on the control signal and previous character states. Similar to other flow-based models, the proposed probabilistic model is designed to approximate the distribution of the current character state.

Our system has three main stages: data preparation, training stage, and runtime stage. During the **data preparation stage**, the training data is prepared, and the control signal parameters are automatically extracted from the data. The user will later provide the control parameters during the runtime stage. During the **training stage** and for each frame of data, the invertible transformations convert the complex character state distribution to a simple base distribution such as a Gaussian distribution with a tractable probability density function. The exact likelihood of each sample $\mathbf{x}_t \in \mathbb{R}^D$ can be evaluated by transforming it to the simple base distribution and then computing the product of the density of the transformed sample ($\mathbf{z}_t \in \mathbb{R}^D$) under the base distribution and the associated change in volume induced by the sequence of invertible transformations. The change in volume is computed by the product of the absolute values of the determinants of the Jacobian

for each transformation, as required by the change of variable formula.

During the **runtime stage** and having optimized model parameters ($\boldsymbol{\theta}$), the invertible transformations transform the simple base distribution $p_Z(\mathbf{z}_t)$ into the complex character state distribution given previous character states and user control. Flowing through a chain of proposed transformations, the variable is substituted by a new one according to the change of variable theorem and eventually, the probability distribution of the character state is obtained. A sample point is drawn from the base distribution at each frame and is mapped to a character state through these transformations. Sampling from base distribution makes the model generate different poses and movements given the same control and history while still following the user control and generating natural and smooth movement variations.

The base of our model is similar to MoGlow [27, 33] with some major improvements as follows:

Architecture: MoGlow uses Glow coupling layers [110] at each frame and conditions the conditioners on one-half of the dimensions in the current frame and also the previous frames to model the spatial and temporal dependencies. Since only one-half of the dimensions are modulated, at least 16 flow steps are needed to have enough expressiveness for transforming the distributions. Inspired by autoregressive flows [181, 182] we propose a model where autoregressive flows across time-steps model the dynamics. In the proposed architecture, we introduce temporal affine layers into each flow-step to modulate all dimensions conditioned on the previous step. In addition, we modulated affine coupling and permutation blocks in each flow step by the previous frames and control signal to improve the modelling of dynamics and have a more responsive character. We will describe each block in detail in Section 6.5. In addition, instead of having a fixed standard normal distribution, we modelled the base distribution as an autoregressive factorization similar to [113] which slightly improved the performance with a negligible additional computational cost. This strategy results in better modelling of temporal dependencies and has significantly improved our results both qualitatively and quantitatively.

Runtime Efficiency: The proposed architecture in MoGlow contains 16 affine coupling steps to be expressive enough. In a normal machine, each forward step takes more than 40 ms, which is equivalent to 25 fps and therefore is not fast enough for real-time game rendering. We exploited several strategies to speed up the runtime process such as 1) Applying linear dimensionality reduction on input/control conditioning data and character state data. This also helped significantly with the

optimization process and a more stable training. 2) Using multi-scale architecture similar to RealNVP [179] and Glow [110]. 3) Applying implementation strategies such as Model Pruning [180]. With the current implementation, each forward step of our network takes around $14ms$ equivalent to 71 fps. If we manage to keep the other computation costs low, our character animation can be rendered at 60 fps in real-time.

Motion Characterization: The input at each frame in MoGlow is only the translational and angular velocity of the root of the character, and the output is the pose of the character resulting in a path following scenario. However, such motion characterization is too simplified and incompatible with the current gaming frameworks. In addition, given such characterization, only the character pose is modelled probabilistically. Our system's input/output data formats follow those of Motion Matching [176]. Using Motion Matching characterization resulted in several advantages and improvements: 1) In this characterization, not only the pose but also the character's movement and the predicted trajectory are modelled probabilistically, delivering a better real-world simulation; 2) This motion characterization is compatible with the gaming frameworks where a gamepad can control the character. In addition, it is compatible with the path following tasks; 3) Modelling joint positions, rotations, and velocities and blending them in the inference achieved much sharper results and was free of bone stretching.

6.4 Preparing Dataset

The first part of this section describes which datasets were used for the project and how the control parameters were extracted. In the second part, we describe the parameterization of the system.

6.4.1 Motion Capture Datasets and Extracting Control Parameters

In this project, we used different sources of motion capture data provided by Holden et al. [3, 4], Starke et al. [31], Ghorbani et al. [161] to train our model. During preprocessing data, the control parameters, including the phase of the gait, frame-wise action labels of the gait, and the trajectory of the root projected on the ground, are manually annotated or computed. In the following, the data preparation and processing steps are described.

6.4.1.1 Unifying Datasets

During data preparation, we converted the selected subsets of data from mentioned datasets into a suitable format for training. First, all of the motion data were retargeted into a uniform skeleton structure with the same bone lengths and a single scale using Autodesk MotionBuilder software [35]. 25 joints represent our character skeleton. Then we mirrored all motion data and doubled the data by augmenting the mirrored versions. The resulting motion data were then subsampled to 60 frames per second. In MoGlow [27], the authors downsampled the data into 20 frames per second to reduce the computational cost and also to make the generation more dependent on the control signals rather than the autoregressive feedback. However, a minimum of 60 frames per second of character control is required in the current gaming platforms. Our design exploited several strategies to reach a low latency for achieving a generation speed of 60 frames per second.

6.4.1.2 Gait Phase

Similar to Holden et al. [4] we also extracted the gait *phase* at each frame of motion data that will be an input parameter to the model for disambiguating the gait cycle during the runtime. We followed the same semi-automatic procedure proposed by Holden et al. [4] to extract phases. First, the foot contact timings were extracted by computing the magnitude of the velocity and also the height of the heel and toe joints and detecting when they fall below a certain threshold. Because this heuristic is prone to failure, the findings were manually examined and adjusted. Given the acquired contact times, the phase was automatically computed as follows: at the frame that the right foot came in contact with the floor phase was set to zero, at the frame that the left foot came in contact with the floor phase was set to π, and at the frame that the right foot came in contact with the floor again phase was set to 2π. The phase for the in-between frames was computed by linear interpolation. The phase for the idle (standing) mode was computed by setting a minimum cycle duration (0.25) where the phase was cycled continuously.

6.4.1.3 Semantic Labels

We also augmented our training dataset with the semantic labels (action types) and converted them to a one-hot encoding. Providing semantic labels significantly helped the model disambiguate the

action types and deliver a better transition between different actions. The influence feeding the semantic labels was significant for challenging situations such as disambiguating fast walking and slow running. In addition to the locomotion labels, semantic labels were necessary to control other actions such as jumping, crouching, or switching the styles. The labels were computed semi-automatically by annotating each action's start and end frame. In addition, a linear interpolation labelled the transition frames between two different actions.

6.4.1.4 Root Motion Trajectory

Providing the future and past root position and orientation of the character has proved to be very effective in controlling the character movement [4, 183, 176, 31, 32]. The root transformation of the character was extracted by projecting the center of the hip joints onto the floor. We examined two definitions for computing the facing direction of the root. In the first definition, the root's facing direction was defined as the $z-axis$ of the center of the hip joints. In the second definition, the facing direction was computed by averaging the vectors between the shoulder and hip joints and taking its cross product with the global $y-axis$ (upward direction). We noticed that the later definition delivered slightly more natural movements. To remove any small, high frequency, or noisy component, the root direction was smoothed over time by a Gaussian kernel of size 20.

6.4.2 Character State, Conditioning Input, and Control Parameters

In this section, we describe the character state, conditioning input, and control parameters. At each frame t, the system requires a set of conditioning input $\mathbf{y}_t$ and control parameters $\mathbf{c}_t$ to modulate the normalizing flows transformations (in both directions). During the runtime, a sample $\mathbf{z}_t$ from base distribution is mapped to the character state $\mathbf{x}_t$ modulated by $\mathbf{y}_t$ and $\mathbf{c}_t$.

Now we describe the details of character state parameters $\mathbf{x}_t$ (See Figure 6.2). Our parameterization is similar to that used in PFNN [4] and Motion Matching [176]. The data contains the root trajectory positions and directions in a window of past and future frames, the annotated action types (labels), and the character joint positions, rotations, and velocities of the current frame. More specifically, for each frame t the character *pose* state is represented by the joint positions, rotations and velocities relative to the root transform. The character *control* state (planning and moving) is represented by root motion and also a surrounding window over root trajectory transformation

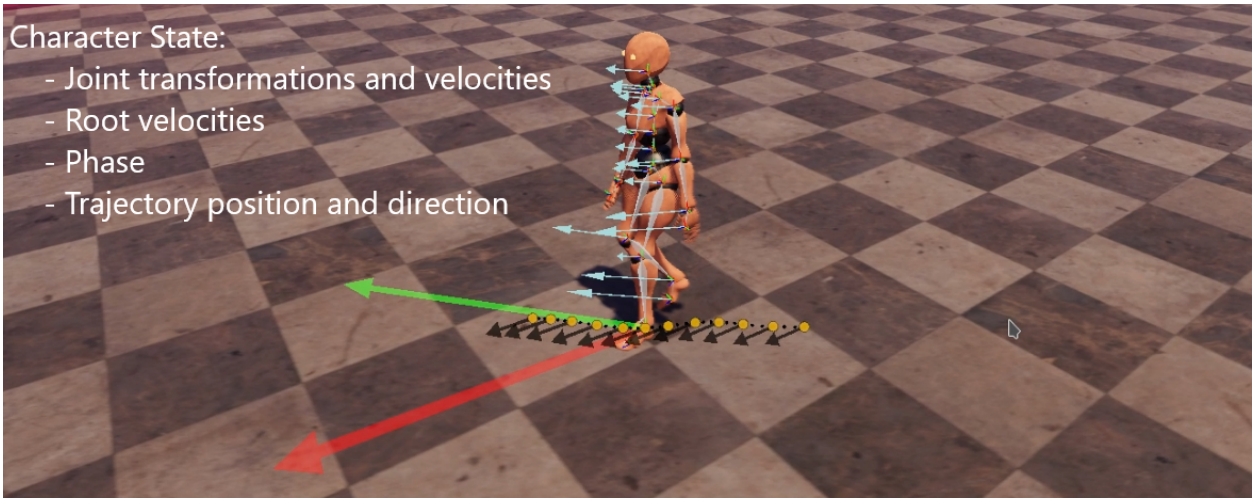


Figure 6.2: A visualization of the character state parameterization of our system. The pose of the character at each time-step is defined by joint transformations in addition to joint velocities (light blue arrows). Sub-sampled predicted trajectory positions and directions are shown in white markers and black arrows, respectively. Gait phase is also predicted at each frame

uniformly sampled at every tenth frame with $n = 12$ samples of surrounding frames. This covers 0.9 seconds of character motion in the future and 1 seconds of character motion in the past. The character state at frame t is defined as $\mathbf{x}_t = \left\{\mathbf{j}^p_t, \mathbf{j}^r_t, \mathbf{j}^v_t, \dot{\mathbf{r}}^x_t, \dot{\mathbf{r}}^z_t, \dot{\mathbf{r}}^a_t, \dot{\mathbf{p}}_t, \mathbf{f}_t, \mathbf{t}^p_{t+1}, \mathbf{t}^d_{t+1}\right\} \in \mathbb{R}^{D_\mathbf{x}}$, where $\mathbf{j}^p_t \in \mathbb{R}^{3j}$ are the joint positions relative to the character root transform, $\mathbf{j}^v_t \in \mathbb{R}^{3j}$ are the joint velocities relative to the character root transform, $\mathbf{j}^r_t \in \mathbb{R}^{6j}$ are the joint rotations relative to the character root transform, $\dot{\mathbf{r}}^x_i \in \mathbb{R}$ and $\dot{\mathbf{r}}^z_i \in \mathbb{R}$ are the root translational velocities relative to the previous frame which defines the translational update of the root, and $\dot{\mathbf{r}}^a_i \in \mathbb{R}$ is the root angular velocity in the 2D-horizontal plane and relative to the orientation of the root at the previous frames which defines the directional update of the root, $\dot{\mathbf{p}}_t \in \mathbb{R}$ is the change in the gait phase, $\mathbf{f}_t \in \{0, 1\}^4$ is the foot contact labels represented as binary variables that indicate if heel or toe joints of the character are in contact with the ground, $\mathbf{t}^p_{t+1} \in \mathbb{R}^{2n}$ are the predicted trajectory positions in the next frame, and $\mathbf{t}^d_{t+1} \in \mathbb{R}^{2n}$ are the predicted trajectory directions in the next frame. Unlike PFNN [4] which used exponential maps, the joint rotations in our system are represented by forward and upward vectors relative to the root transform to avoid quaternion interpolation issues which usually happens during training the neural networks. The rotations are transformed back to quaternions during runtime.

The conditioning input at frame t is defined as $\mathbf{y}_t = \left\{\mathbf{j}^p_{t-\tau:t-1}, \mathbf{j}^r_{t-\tau:t-1}, \mathbf{j}^v_{t-\tau:t-1}, \mathbf{p}_t\right\} \in \mathbb{R}^{D_\mathbf{y}}$, where $\mathbf{j}^p_{t-\tau:t-1} \in \mathbb{R}^{3j\tau}$ is the history of local joint positions for τ steps, $\mathbf{j}^r_{t-\tau:t-1} \in \mathbb{R}^{6j\tau}$ is the history

of local joint rotations for τ steps, $\mathbf{j}^v_{t-\tau:t-1} \in \mathbb{R}^{3j\tau}$ is the history of local joint velocities for τ steps, and $\mathbf{p}_t = \mathbf{p}_{t-1} + \dot{\mathbf{p}}_t$ is the current phase of the gait. The control signal at frame t is defined as $\mathbf{c}_t = \left\{\mathbf{t}^p_t, \mathbf{t}^d_t, \mathbf{t}^g_t\right\} \in \mathbb{R}^{D\mathbf{c}}$, where $\mathbf{t}^p_t \in \mathbb{R}^{2n}$ are the subsampled trajectory positions, $\mathbf{t}^d_t \in \mathbb{R}^{2n}$ are the subsampled trajectory rotations, and $\mathbf{t}^g_t \in \mathbb{R}^{mn}$ are the one-hot encoding of the gait action types (labels) at the subsampled frames. We reduced the dimensionality of modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$, and character state $\mathbf{x}_t$ by PCA which significantly improved the model efficiency during both training and runtime stages, and without loosing any perceptible variation in the synthesized motion.

6.5 KinFlow Architecture

In this section, we describe KinFlow and its components. KinFlow is a recursive normalizing flows model constructed by a sequence of modulated invertible transformations that model the dependencies within the character's very high-dimensional motion, which can be specified in the form of a full joint probability distribution. The invertible transformation is modulated by control signals and previous information to follow the user control and model the temporal dependencies.

The probability density function of the character state at each frame, $\mathbf{x}_t \in \mathbb{R}^D$, is computed as follows:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{c}_t) &= p(\mathbf{z}_t) \left|\det\left(\frac{\partial \mathbf{x}_t}{\partial \mathbf{z}_t}\right)\right|^{-1} \\ &= p_Z\left(\mathbf{g}^{-1}_{\boldsymbol{\theta}}(\mathbf{x}_t; \mathbf{y}_t, \mathbf{c}_t)\right) \left|\det\left(\frac{\partial \mathbf{x}_t}{\partial \mathbf{z}_t}\right)\right|^{-1} \\ &= p_Z\left(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_t; \mathbf{y}_t, \mathbf{c}_t)\right) \left|\det\left(\frac{\partial \mathbf{z}_t}{\partial \mathbf{x}_t}\right)\right|, \end{aligned} \tag{6.1}$$

$$\log(p_{\boldsymbol{\theta}}(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{c}_t)) = \log\left(p_Z\left(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_t; \mathbf{y}_t, \mathbf{c}_t)\right)\right) \log\left(\left|\det\left(\frac{\partial \mathbf{z}_t}{\partial \mathbf{x}_t}\right)\right|\right), \tag{6.2}$$

where $p_Z(\mathbf{z}_t)$ is a simple distribution (in our case Gaussian), $\mathbf{f}_{\boldsymbol{\theta}}$ represents the stack of all invertible transformations parameterized by $\boldsymbol{\theta}$ that maps $\mathbf{x}_t$ to $\mathbf{z}_t$, $\mathbf{g}_{\boldsymbol{\theta}}$ is the inverse of $\mathbf{f}_{\boldsymbol{\theta}}$, $\frac{\partial \mathbf{x}_t}{\partial \mathbf{z}_t}$ and $\frac{\partial \mathbf{z}_t}{\partial \mathbf{x}_t}$ are the Jacobian of functions $\mathbf{g}_{\boldsymbol{\theta}}$ and $\mathbf{f}_{\boldsymbol{\theta}}$, respectively, $\mathbf{c}_t$ is the user control, and $\mathbf{y}_t = \mathbf{h}(\mathbf{x}_{<t})$ is the conditioning input as a function of the history of previous character states for the current frame (see Figure 6.1). To model the temporal dependencies and control conditioning, the invertible transformations are modulated by user control $\mathbf{c}_t$ and previous states $\mathbf{x}_{<t}$. Later, we will explain how each transformation is modulated by user control and previous states.

The flow structure of our model is constructed by a sequence of Flow-Steps similar to the model proposed in [110] as follows:

$$\mathbf{x}_t \xleftrightarrow{\mathbf{f}_1} \mathbf{z}_t^1 \xleftrightarrow{\mathbf{f}_2} \mathbf{z}_t^2 \dots \xleftrightarrow{\mathbf{f}_L} \mathbf{z}_t^L = \mathbf{z}_t, \tag{6.3}$$

where $\mathbf{f}_l$ represents one Flow-Step. The main difference between the Flow-Steps proposed in Glow [110] and MoGlow [27] is that, 1) all modules except Actnorm in our proposed Flow-Step are conditioned on previous states and control signals, 2) we added a proposed temporal affine layer into Flow-Steps to improve the modelling of motion dynamics. Each Flow-Step in our model can be represented as:

$$\mathbf{f}_l = \mathbf{TA}_l \circ \mathbf{C}_l \circ \mathbf{P}_l \circ \mathbf{N}_l, \tag{6.4}$$

where $\mathbf{TA}_l$ is an affine temporal layer, $\mathbf{C}_l$ is the coupling layer, $\mathbf{P}_l$ is the 1×1 convolution layer, and $\mathbf{N}_l$ is the normalization layer. In the following we will explain each layer of the proposed Flow-Step in more detail.

6.5.1 Normalization Layer

In the normalization layer we use the *actnorm* layer (short for activation normalization layer) which was proposed in Glow [110]. *Actnorm* is parameterized by a scale $\boldsymbol{\gamma} \in \mathbb{R}^D$ and bias $\boldsymbol{\beta} \in \mathbb{R}^D$ and performs an affine transformation on each channel of the activations. The actnorm function, its inverse, and its log-determinant is defined as follows:

$$z_t = \boldsymbol{\gamma} \odot x_t + \boldsymbol{\beta}, \tag{6.5}$$

$$x_t = (z_t - \boldsymbol{\beta}) / \boldsymbol{\gamma}, \tag{6.6}$$

$$\log \left(\left| \det \left(\frac{\partial z_t}{\partial x_t} \right) \right| \right) = \text{sum}(\log |\boldsymbol{\gamma}|), \tag{6.7}$$

where $x_t \in \mathbb{R}^D$ and $z_t \in \mathbb{R}^D$ represent the input and output of this layer layer at frame t and $\odot$ is the element-wise multiplication. Actnorm was proposed as a substitute to batch normalization to address the problem of introduced noise when the minibatch size per GPU is small. Scale $\boldsymbol{\gamma}$ and bias $\boldsymbol{\beta}$ are trainable parameters and are initialized so that the resulting activations of Actnorm for

the first mini-batch have zero mean and unit variance for each channel.

6.5.2 1×1 Convolution Layer

As a generalization of the permutation layer, we propose an extension of Invertible 1×1 Convolution that was originally proposed in Glow [110]. Invertible 1×1 Convolution was proposed as a replacement for the fixed permutation used in RealNVP [179] which was introduced to permute the order of the channels. Invertible 1×1 Convolution applies a channel-wise linear transformation via a learnable weight matrix $\mathbf{W} \in \mathbb{R}^{D \times D}$ as a generalization of permutation operation. Invertible 1×1 Convolution, its inverse, and its log-determinant is defined as follows:

$$z_t = \mathbf{W} x_t, \tag{6.8}$$

$$x_t = \mathbf{W}^{-1} z_t, \tag{6.9}$$

$$\log\left(\left|\det\left(\frac{\partial z_t}{\partial x_t}\right)\right|\right) = \log|\det(\mathbf{W})|. \tag{6.10}$$

The weight matrix $\mathbf{W}$ is initialized to a random rotation matrix with a log-determinant of zero, which initially diverges during the gradient decent process.

The computational cost of $\det(\mathbf{W})$ is $\mathcal{O}\left(D^3\right)$. The authors of Glow also proposed using **LU Decomposition** to reduce the computational complexity of $\det(\mathbf{W})$ from $\mathcal{O}\left(D^3\right)$ to $\mathcal{O}\left(c\right)$ which can be significant for large number of channels D. Weight matrix $\mathbf{W}$ is parameterized by LU decomposition as follows:

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \text{diag}(\mathbf{s})), \tag{6.11}$$

where $\mathbf{P} \in \mathbb{R}^{D \times D}$ is a fixed permutation matrix, $\mathbf{L} \in \mathbb{R}^{D \times D}$ is a learnable lower triangular matrix with ones on its diagonal, $\mathbf{U} \in \mathbb{R}^{D \times D}$ is a learnable upper triangular matrix with zeros on its diagonal, and $\mathbf{s} \in \mathbb{R}^{D \times D}$ is a learnable vector. The log-determinant in this case is

$$\log|\det(\mathbf{W})| = \text{sum}(\log|\mathbf{s}|). \tag{6.12}$$

Using LU decomposition was an essential requirement in this project since the number of channels ($D = D_x$) was very high.

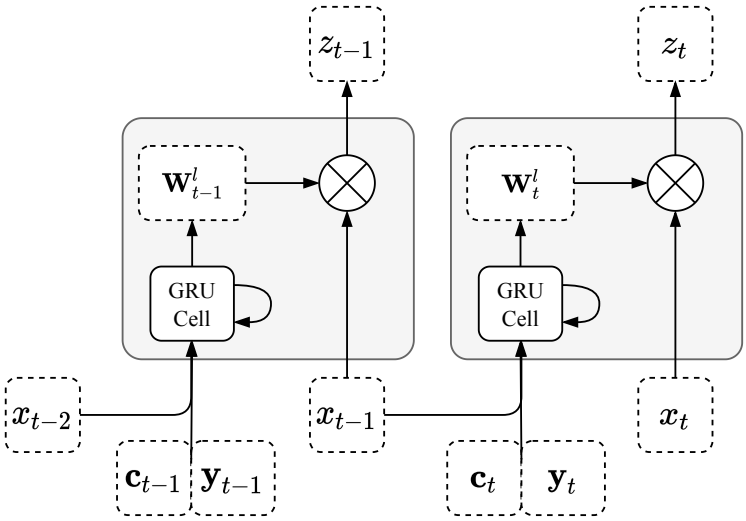


Figure 6.3: The proposed 1×1 convolution layer. At each step the weight matrix for 1×1 convolution is computed as a function of modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$ and the previous input the this layer x_t.

In this work we propose an extension of Invertible 1×1 Convolution for a better modelling of dynamics where the weight matrix $\mathbf{W}$ is a function of modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$. Each step in the proposed 1×1 convolution layer is performed as follows:

$$(\mathbf{L}^l_t, \mathbf{U}^l_t, \mathbf{s}^l_t) = f^l_{\mathbf{P}}(\mathbf{c}_t, \mathbf{y}_t, x_{<t}), \tag{6.13}$$

$$\mathbf{W}^l_t = \mathbf{P}\mathbf{L}^l_t(\mathbf{U}^l_t + \text{diag}(\mathbf{s}^l_t)), \tag{6.14}$$

$$z_t = \mathbf{W}^l_t x_t, \tag{6.15}$$

where $\mathbf{L}^l_t, \mathbf{U}^l_t, \mathbf{s}^l_t$ are the LU decomposition parameters of weight matrix $\mathbf{W}^l_t$ for flow-step l at frame t, and x_{t-1} is the input to this layer at the previous frame. $f^l_{\mathbf{P}}$ is an arbitrary complex function such as a neural network. In our work we used Gated Recurrent Unit (GRU) which is capable of modelling long-term dependencies.

$$\begin{aligned} \mathbf{h}_{t,l} &= \text{GRUCell}\left((\mathbf{c}_t, \mathbf{y}_t, x_{t-1}), \mathbf{h}_{t-1,l}\right), \\ (\mathbf{L}^l_t, \mathbf{U}^l_t, \mathbf{s}^l_t) &= \text{Linear}\left(\mathbf{h}_{t,l}\right). \end{aligned} \tag{6.16}$$

Figure 6.3 illustrates the 1×1 convolution layer in our model.

6.5.3 Coupling Layer

For the coupling layer we used the same architecture used in MoGlow [27] which was an extension of affine coupling layer proposed in RealNVP [179]. Forward direction, inverse direction, and log-determinant of the used coupling layer can be expressed as:

$$\begin{aligned} z_t^{1:d} &= x_t^{1:d}, \\ z_t^{d+1:D} &= x_t^{d+1:D} \odot \exp\left(\mathbf{\Theta}_1\right) + \mathbf{\Theta}_2, \end{aligned} \tag{6.17}$$

$$\begin{aligned} x_t^{1:d} &= z_t^{1:d}, \\ x_t^{d+1:D} &= (z_t^{d+1:D} - \Theta_2) \odot \exp\left(-\Theta_1\right), \end{aligned} \tag{6.18}$$

$$\log\left(\left|\det\left(\frac{\partial z_t}{\partial x_t}\right)\right|\right) = \sum_j \Theta_{1,i}, \tag{6.19}$$

where Θ_1 and Θ_2 are functions of $x_{\leq t}^{1:d}$ and the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$.

$$(\Theta_1, \Theta_2) = f_{\mathbf{C}}^l\left(\mathbf{c}_t, \mathbf{y}_t, x_{\leq t}^{1:d}\right), \tag{6.20}$$

where $f_{\mathbf{C}}^l$ is a complex function such as a neural network. Similar to the 1×1 convolution layer, we used a GRU Cell to represent $f_{\mathbf{C}}^l$:

$$\begin{aligned} \mathbf{h}_{t,l} &= \text{GRUCell}\left(\left(\mathbf{c}_t, \mathbf{y}_t, x_t^{1:d}\right), \mathbf{h}_{t-1,l}\right), \\ (\Theta_1, \Theta_2) &= \text{Linear}\left(\mathbf{h}_{t,l}\right), \end{aligned} \tag{6.21}$$

Figure 6.4 illustrates the structure of the coupling layer in our architecture.

We also examined using **Continuous Mixture CDFs** coupling layers which was originally proposed in **Flow++** model [184] where the forward direction can be defined as follows:

$$\begin{aligned} z_t^{1:d} &= x_t^{1:d}, \\ z_t^{d+1:D} &= F(x_t^{d+1:D}; \mathbf{\Theta}_3) \odot \mathbf{\Theta}_1 + \mathbf{\Theta}_2, \end{aligned} \tag{6.22}$$

where Θ_1, Θ_2 and $\Theta_3 = [\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{s}]$ are functions of $x_{\leq t}^{1:d}$ and the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$. The function F is a cumulative distribution function (CDF) constructed by a mixture of K logistics

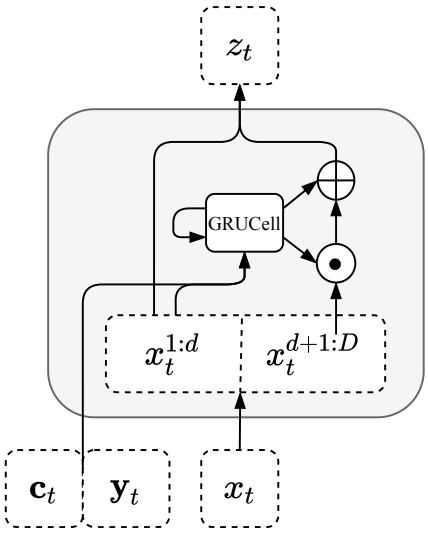


Figure 6.4: The coupling flow layer is used in our model. We modelled the conditioner using a GRU Cell and conditioned that to the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$ in addition to the first data partition $x_t^{1:d}$

followed by an inverse sigmoid.

$$F(x, \pi, \boldsymbol{\mu}, \mathbf{s}) = \sigma^{-1} \left(\sum_{j=1}^{K} \pi_j \sigma \left(\frac{x - \mu_j}{s_j} \right) \right). \tag{6.23}$$

The inverse of 6.22 is computed numerically using a bisection algorithm.

This slightly improved our results in terms of log-likelihood with the cost of more training time. However, since we did not notice any difference in the quality of generated motions, we used affine coupling layers for reporting the results.

6.5.4 Temporal Affine Layer

Inspired by autoregressive flows [181, 182], we introduced another layer to each flow step to improve the modelling of human motion dynamics in our architecture. In this layer, which we call **Temporal Affine Layer** dynamics are modelled by autoregressive flows across time-steps. The autoregressive transformation at each time step is defined as follows:

$$z_t = \mathbf{TA}_l(x_t; \Theta(x_{<t})), \tag{6.24}$$

where $\mathbf{TA}_l(\cdot;\theta) : \mathbb{R}^D \to \mathbb{R}^D$ is a bijection parameterized by θ and Θ that is called the *conditioner*. In our work we used an affine autoregressive transformation defined as follows:

$$z_t = \boldsymbol{t} + \exp(\boldsymbol{s}) \odot x_t, \tag{6.25}$$

where $\boldsymbol{t}$ and $\boldsymbol{s}$ are the conditioner functions of $x_{<t}$ and the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$:

$$(\boldsymbol{t}, \boldsymbol{s}) = f^l_{\mathbf{TA}}\left(\mathbf{c}_t, \mathbf{y}_t, x_{<t}\right), \tag{6.26}$$

where $f^l_{\mathbf{TA}}$ is a complex function such as a neural network. Similar to the permutation and coupling layers, we used a GRU Cell to represent $f^l_{\mathbf{TA}}$:

$$\begin{aligned} \mathbf{h}_{t,l} &= \text{GRUCell}((\mathbf{c}_t, \mathbf{y}_t, x_{t-1}), \mathbf{h}_{t-1,l}), \\ (\mathbf{t}, \mathbf{s}) &= \text{Linear}(\mathbf{h}_{t,l}). \end{aligned} \tag{6.27}$$

The Jacobian matrix is in a triangular form due to the autoregressive structure of our bijection. The log determinant of Jacobian for each time-step is computed as follows:

$$\log \left| \det \left(\frac{\partial z_t}{\partial x_t} \right) \right| = \sum_{i=1}^{D} \boldsymbol{s}_i, \tag{6.28}$$

and the inverse transformation is defined as:

$$x_t = (z_t - \boldsymbol{t}) \odot \exp(-\boldsymbol{s}). \tag{6.29}$$

The temporal affine layer can be seen as an extension of the Actnorm layer with the difference that the scaler and bias are conditioned on the previous frames and the modulating vector. We can also consider the temporal affine layer as a special case of coupling layer where one partition is constructed by $x_{<t}$ and the other partition is constructed by x_t. Figure 6.5 shows the architecture of the temporal affine layer.

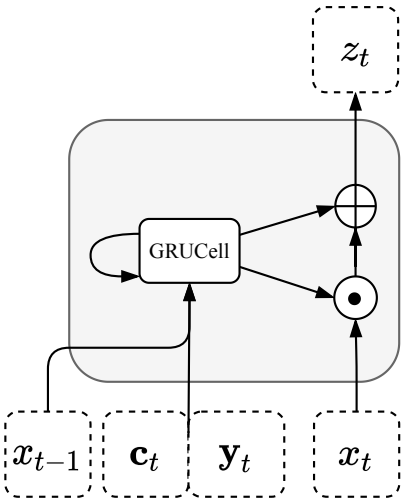


Figure 6.5: The temporal affine layer is used in our model. The input is modulated by the modulating vector $[\mathbf{c}_t; \mathbf{y}_t]$ and the input in the previous frames $x_{<t}$ using a GRU.

6.5.5 Multi-scale Architecture

To reduce the computational and memory cost of the model, we followed the multi-scale architectural design choice proposed in RealNVP [109, 110] and factored out half of the dimension after applying half of the flow steps. The factored out dimensions were Gaussianized. This strategy decreased the number of trainable parameters effectively and increased the speed of our model during the training and runtime.

6.5.6 Objective function

Our model is trained by maximizing the log-likelihood objective function over all frames in our training dataset $\mathcal{D}$ as follows:

$$\begin{aligned} \log p(\mathcal{D} \mid \theta) &= \max_{\boldsymbol{\theta}} \sum_{i=1}^{M} \sum_{t} \log p_{\boldsymbol{\theta}} \left(\mathbf{x}_t^{(i)} \mid \mathbf{x}_{<t}^{(i)} \mathbf{c}_t^{(i)} \right) \\ &= \max_{\boldsymbol{\theta}} \sum_{i=1}^{M} \sum_{t} \log p_Z \left(\mathbf{f}_\theta \left(\mathbf{x}_t^{(i)}; \mathbf{y}_t^{(i)} \mathbf{c}_t^{(i)} \right) \right) + \log \left| \det \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \mathbf{x}_t^{(i)}} \left(\mathbf{x}_t^{(i)} \right) \right|, \end{aligned} \tag{6.30}$$

where $M = \mid \mathcal{D} \mid$ is the number of elements in the training dataset. $\mathbf{z}_t^{(i)} = \mathbf{f}_\theta \left(\mathbf{x}_t^{(i)}; \mathbf{y}_t^{(i)} \mathbf{c}_t^{(i)} \right)$ can be computed by computing the chain of flow steps $\mathbf{f}_\theta = \mathbf{f}_1 \circ \mathbf{f}_2 \circ \cdots \circ \mathbf{f}_L$. Also the log-determinant part can be computed by summing over the log-determinant of all transformations.

6.6 Implementation and Training

6.6.1 Data processing

We sliced all motion sequences in the dataset into the fixed-length windows of 4 seconds (240 frames) with an overlap of 2 seconds (120 frames) where each frame of data given to the model is represented by character state $\mathbf{x}_t \in \mathbb{R}^{316}$ and modulating vector $[\mathbf{c}_t; \mathbf{y}_t] \in \mathbb{R}^{1561}$. We tested a different number of autoregressive time-steps τ by visual investigation of output results and set it to 5 as we did not get any noticeable change for higher numbers. We normalized the data using mean values $(\mathbf{x}_\mu, \mathbf{y}_\mu, \mathbf{c}_\mu)$ and standard deviations $(\mathbf{x}_\sigma, \mathbf{y}_\sigma, \mathbf{c}_\sigma)$ following by a scaling step using weight vectors $(\mathbf{x}_w, \mathbf{y}_w, \mathbf{c}_w)$ which determines the relative importance of each dimension. We used a similar scaling scheme to PFNN [4] by scaling the joint data in the conditioning input (positions, rotations, and velocities) by a value of 0.1 to reduce their importance and increase the reliance of generated motion on the control signal (trajectory data and semantics labels). With similar motivation, MoGlow used a data dropout layer on the autoregressive history of inputs (which is similar to our conditioning input $\mathbf{y}_t$ to some extent) to increase the adherence to the control input and address the problem of *over-reliance* on autoregressive pose information. We also explored the idea of data dropout but achieved a relatively better result by scaling the conditioning input without any dropout. In the next step, we applied a PCA dimensionality reduction on character state data and modulating vector data (see Figure 6.1). The number of components for both PCA blocks was selected such that the amount of variance that was needed to be explained was greater than 0.99 of the total variance. After applying PCA dimensionality reduction, the number of dimension for character state $\mathbf{x}_t$ and modulating vector and $[\mathbf{c}_t; \mathbf{y}_t]$ were reduced to 74 and 77, respectively. Using PCA significantly reduced the number of dimensions in the data resulting in a much shorter training time and faster execution during the runtime. Using PCA also has the advantage of removing the noise in the motion capture data. Finally, we took out a small proportion of the dataset (5 percent) as the validation dataset.

6.6.2 Model Parameters

We use a multi-scale architecture similar to [109], and [110] where half of the dimensions are factored out after 4 flow-steps, and then we have another 4 flow-steps on the remaining dimensions. This architecture makes our model much faster compared to MoGlow with 16 flow-steps. We used GRU cells with a hidden state size of 128 for all permutation, coupling, temporal affine, and base distribution layers, followed by a linear layer.

6.6.3 Training Process

For training KinFlow, we optimized the objective function in Eq. 6.30 using a stochastic gradient descent Adam optimizer [170] and starting with a learning rate of 0.002 with a multiplicative decay of 0.95 for each 20 epochs. We set the gradient norm clipping to 0.1 to avoid any exploding gradients. We also used a mixed-precision strategy which reduced the memory footprint during model training and improved performance by achieving $1.5X$ speedup on GPU. The initial hidden states for all GRUs were set to zeros. The training was performed for $120k$ iteration with mini-batches of size 100, and took around 13 hours on an NVIDIA GeForce RTX 2080 Ti. The model was implemented in PyTorch [185, 186].

6.6.4 Runtime

During runtime and at each frame, the mean and standard deviation of the base distribution is computed given the history of these parameters in the previous frames. Then, a point $\mathbf{z}_t$ is sampled from the computed base distribution and transformed to the character state $\mathbf{x}_t$ by applying the sequence of transformations (6.3).

Given the character state $\mathbf{x}_t$ the root and joint transformations can be computed for rendering the character pose at frame t. The root position and rotation are updated using predicted root translational and rotational velocities $\{\dot{\mathbf{r}}^x_t, \dot{\mathbf{r}}^z_t, \dot{\mathbf{r}}^a_t\}$. The joint transformations are computed from from the predicted joint positions, angles, and velocities $\{\mathbf{j}^p_t, \mathbf{j}^r_t, \mathbf{j}^v_t\}$. The character joint positions were computed by averaging the predicted joint position and the previous joint positions added by the predicted joint velocities.

The joint components of conditional input ($\{\mathbf{j}^p_{t-\tau:t-1}, \mathbf{j}^r_{t-\tau:t-1}, \mathbf{j}^v_{t-\tau:t-1}\}$) are updated autore-

gressively by the predicted joint positions, rotations, and velocities. The phase can also be updated by adding the predicted change in phase ($\mathbf{p}_t = \mathbf{p}_{t-1} + \dot{\mathbf{p}}_t$).

For controlling the character, we utilized the same strategy proposed in PFNN [4]. The position and direction of future points in the trajectory (which are used in control signal $\mathbf{c}_t$) were computed by blending the trajectory predicted by the model and the desired facing and moving (scaled by speed) direction given by the game-pad control as follows:

$$\mathbf{p}[i] = (1 - (i/N)^{\tau})\, \mathbf{p}_{control}[i] + (i/N)^{\tau} \mathbf{p}_{predicted}[i], \tag{6.31}$$

where $\mathbf{p}[i]$ represents the position or direction of the ith future point in the trajectory and N is the total number of future points in the trajectory. τ represents a bias for tuning the amount of character responsiveness to the game-pad. A higher τ results in a blending function that biases towards the predicted values by the network. This generates more natural motion but is less responsive to the game-pad. On the other hand, a lower τ results in a higher bias towards the game-pad, increasing the responsiveness to user control. We set the bias for positions to $\tau_p = 1.5$ and the bias for directions to $\tau_d = 0.75$. The action types $\mathbf{t}_t^g$ were also given directly from the game-pad.

Rendering of the animations was implemented in Unity 3D, which was fed by the data coming from the python code through a socket terminal.

6.7 Results

6.7.1 Experiments

In this section, we show the results generated by our model in a number of different situations and tasks. Please also refer to the supplemental video for additional visual results. All results are shown using the default setting described in Section 6.6.

Figure 6.6, 6.7, and 6.8 show the left foot height, left arm rotation, and the projected root onto the ground for two samples generated from the exact same input control. The results show that our model produces different outputs given the same input while still following the control commands.

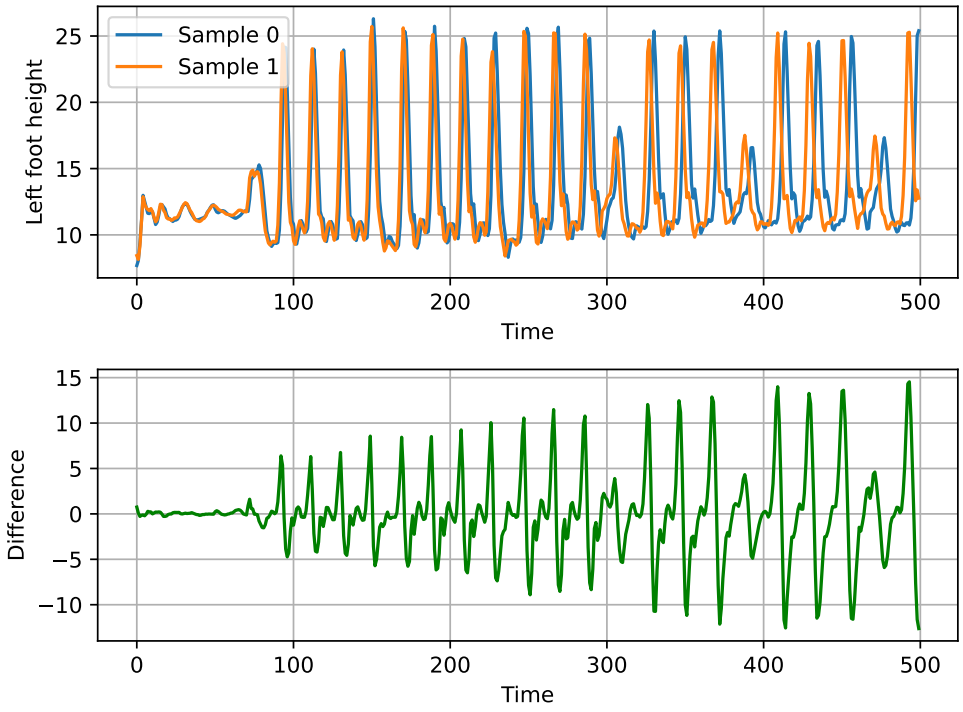


Figure 6.6: Top: Height of the left foot (in centimetres) for two generated samples from the exact same input control. Bottom: The difference between the left foot heights of the two samples.

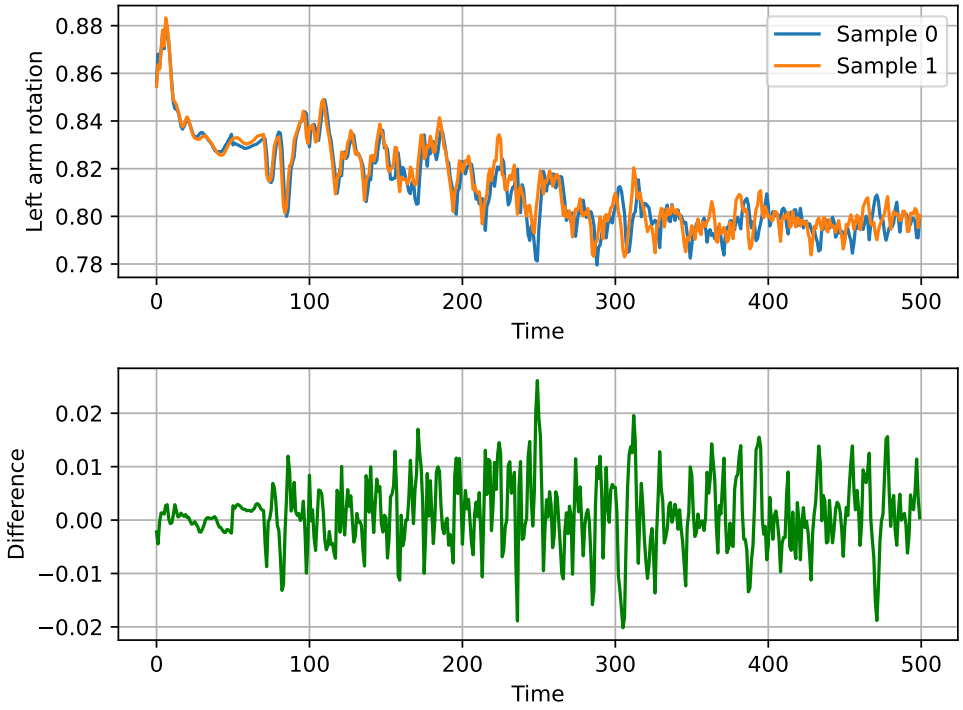


Figure 6.7: Top: The real part of the left arm rotation for two generated samples from the exact same input control. Bottom: The difference between the left arm rotations of the two samples.

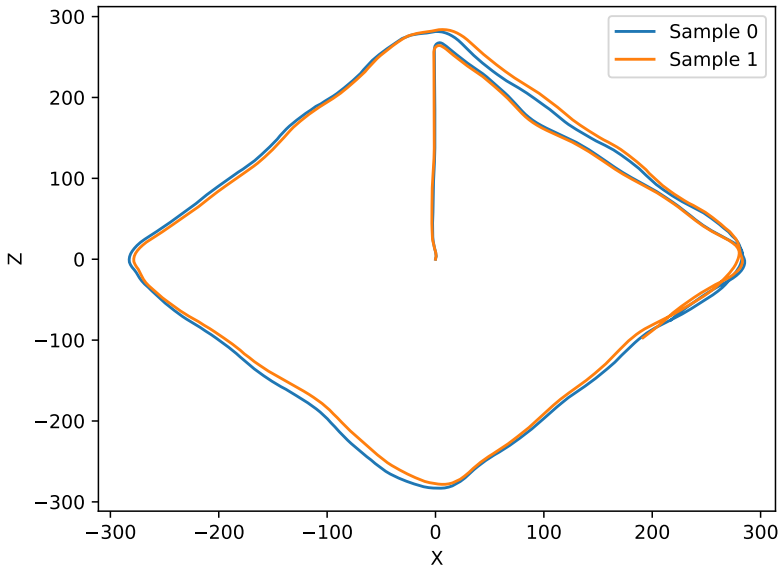


Figure 6.8: Projected root onto the horizontal plane for two generated samples from the exact same input control.

Figure 6.9: Results showing our character crouching

In Figures 6.9 and 6.10 we show our method can produce different action types such as jumping or crouching controlled by the user. Moreover, our character produces realistic and responsive movements for each action type.

In Figure 6.11, we show how the character navigates in the environment performing tight or lose turns, lateral side waking and running, changes in speed, and different face directions. The character is responsive to the user input while producing convincing and realistic motion for different inputs. In addition, our probabilistic approach encodes the stochastic nature of movement in the produced motions. Given the exact same user control, various joint movement trajectories are generated while still responding to the user control. Furthermore, we can control the amount of injected stochasticity by changing the temperature when sampling from the base distribution [110].

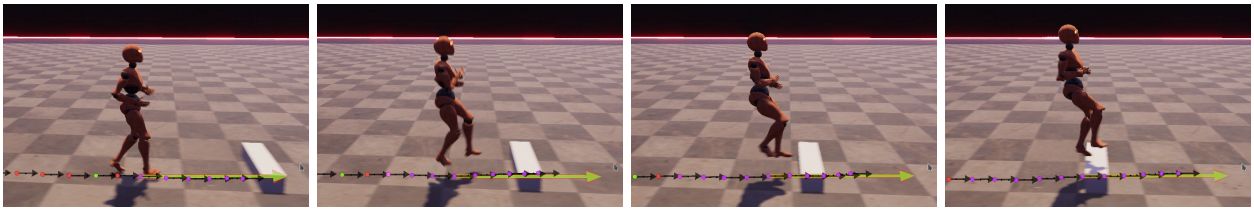
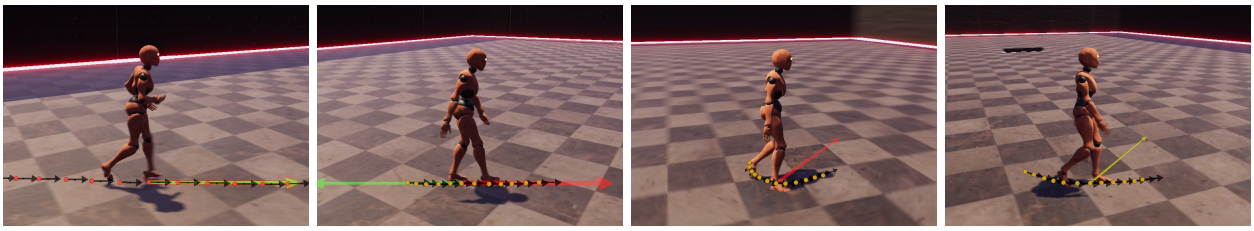
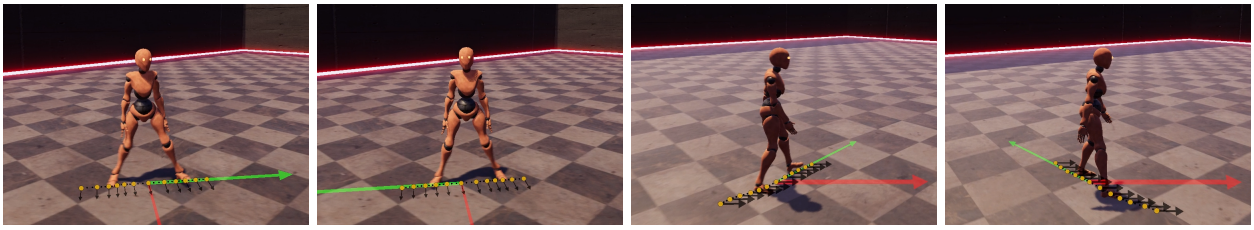


Figure 6.10: Results showing our character jumping.



(a) Running **(b)** Walking backward **(c)** Tight turn **(d)** Loose turn



(e) Side walk (left) **(f)** Side walk (right) **(g)** Forward + left **(h)** Backward + left

Figure 6.11: Results showing our character controlled by adjusting the future trajectory position and direction and action type.

Our character can also follow a predefined path on the ground. Figure 6.12 shows how our character follows different predefined paths on the ground. The responsiveness of the character can be adjusted by the bias in 6.31 for movement and facing direction.

We also adopted a similar approach as PFNN [4] for avoiding walls and other untraversable obstacles where the projected trajectory in the future detects the potential obstacle and slows down the character. We categorized objects into obstacles/non-obstacles by baking a Navmesh from the level geometry in the Unity. Figure 6.13 shows our character moving in walkable areas and avoiding the obstacles.

6.7.2 Quantitative Evaluation

For the purpose of quantitative comparison, we compared our model with MoGlow, which is the closest model to ours in terms of design. MoGlow adopted a simplistic path following the control

Figure 6.12: Controlling our character by predefined paths.

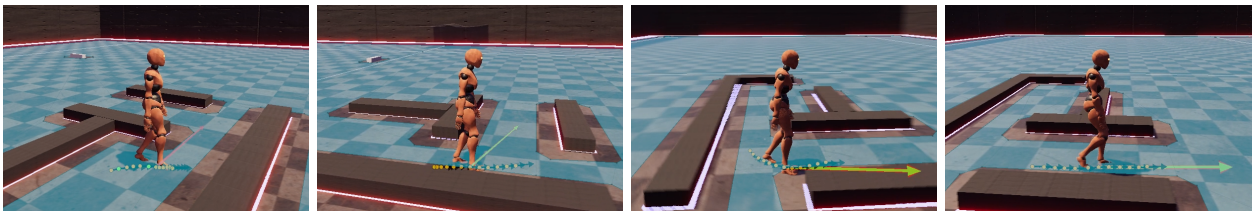


Figure 6.13: Samples of character moving in the walkable areas. The environment is baked for categorizing objects using Unity Navmesh for obstacle avoidance.

paradigm where the character follows a predefined path on the ground. For the purpose of comparison, we changed their control representation to our representation. In addition, in MoGlow the pose is represented only by joint positions or joint angles. Therefore, we also changed it to our pose representation (positions, rotations, velocities). We used the same network parameters as proposed in the original work. We denote MoGlow by **MG1** in our results. MoGlow can be seen as one of our ablations where the temporal dependencies are modelled only in coupling flows. Therefore, in addition to MoGlow with original network parameters, we trained our designed model where the temporal dependencies are only modelled in coupling flows by using simple 1×1 convolution layers, removing affine temporal layers, and using not recurrent modelling in the base distribution. We denote this version by **MG2** in our results. We also trained four additional ablated configurations of our model to evaluate the contribution of modelling dynamics in each layer. In the first ablated configuration, we replace the temporal 1×1 convolution layer with the non-temporal original version (similar to Glow and MoGlow) labelled **NoTempPerm**. In the second ablated configuration, we replaced the recurrent architecture of the coupling layer with a 3-layer MLP which is not conditioned on the conditioning input. This model is denoted by **NoTempCoup**. We removed the temporal affine layer from our network for the third ablated configuration. This model is denoted by **NoTempAffine**. For the third ablated configuration, we model the base distribution by an unconditioned standard Gaussian instead of modelling it recursively and conditioned on the

Model	NLL ↓	Training Time ↓	Num. of Params. ↓	Runtime Speed ↑
MG1	−7.35	15.6 h	36.93 M	23 fps
MG2	−4.11	9.81 h	5.86 M	81 fps
NoTempCoup	−7.12	11.61 h	17.9 M	71 fps
NoTempPerm	−7.91	11.78 h	17.6 M	71 fps
NoTempAffine	−7.43	11.31 h	17.6 M	75 fps
NoTempBase	−8.11	12.97 h	18.1 M	67 fps
Full	−8.28	13.12 h	18.5 M	66 fps

Table 6.1: Comparing different models for modelling motion data in terms of negative-log-likelihood (NLL) per dimension on the held-out test data, training duration, the number of learnable parameters, and runtime speech.

base distribution in previous steps. We denote this model by **NoTempBase**. We denote the final proposed model by **Full**.

The results of the quantitative evaluation are shown in Table 6.1. Our primary evaluation metric is defined as negative- log-likelihood (NLL) per dimension on the held-out test data, which shows how well our model fits to the test data. In addition to NLL, we report the overall training time for the same number of training steps (160K), the number of learnable parameters, and also the runtime speed as the number of generated frames per second.

Modelling temporal dynamics only in coupling layers and when we used our shallower multi-scale structure with smaller hidden states for the GRU cells (**MG2**) achieved the highest NLL. The original MoGlow achieved relatively low NLL; however, the training time and the number of learnable parameters are much higher than in other models. More importantly, it takes around 43.4 milliseconds (23 fps) for this model to process one frame, which is much slower than 60 fps needed for common gaming platforms. This is due to the higher number of flow-steps and larger GRU cells (two layers of GRU with a hidden state size of 512) for each coupling layer. By comparing the NLL in test dataset for ablated configurations, **NoTempPerm**, **NoTempCoup**, **NoTempAffine**, and **NotRecBase**, we can conclude that the coupling layer has the highest contribution in modelling dynamics in our F**ull** architecture followed by our proposed Temporal Affine layer. In these models, **NoTempAffine** achieved the highest runtime speed (9 fps higher than **Full** model), showing the influence of the Temporal Affine Layer in slowing down our model. Among all models, the **Full** proposed parameterization achieved the best results in terms of NLL. This model achieves a reasonable runtime speed of 66 fps.

6.8 Conclusion

In this work, we proposed a probabilistic model for character control as an extension previous flow-based model MoGlow [27] which achieves lower negative-log likelihood per dimension, higher runtime speed enough for the common gaming platforms, and is compatible with the common user control inputs. In this work, we proposed an additional layer for modelling motion dynamics in flow transformation, which significantly improved our results. In addition, we exploited several strategies to speed up the runtime process achieving 60 fps while not sacrificing the quality of the generated motion. Furthermore, since we are characterizing the input/output of our model similar to those of Motion Matching [17], not only pose but also the character's movement trajectory is also modelled probabilistically, resulting in more natural and realistic character movements. In addition, such characterization is more compatible with the current gaming interfaces.

6.9 Future Work

Our framework was proposed to model character control for locomotion and minimal interaction with the environment (obstacle vs non-obstacle). This was feasible as we do not need in-frame responsiveness to the user control in locomotion. Adding other types of movements such as sitting or lifting entails an in-frame response to the user control and interaction with the environment. However, in general, approaches based on recurrent neural networks showed to be slow in such situations as their hidden states cannot be quickly updated to another subspace of the motion. As opposed to such models, autoregressive models which use Mixture-of-Experts architectures [12, 31, 32] to divide the motion space into homogeneous regions showed higher responsiveness in updating the action type.

Although we managed to speed up our framework to higher than 60 fps, it is still slower than the memoryless autoregressive models. This is an essential factor when many resources are required for other parts of the game, such as rendering the environment, animating the Non-Player Characters (NPC), or interacting with the environment.

Our proposed framework was designed for a flat plane. One of the main future works is to extend the current framework for moving the character on uneven terrain. This can be done by

training the model on a dataset with uneven terrain and providing the height of the surrounding area as an additional input to the model.

Chapter 7

Conclusions and Future Directions

7.1 Summary of Contributions

This dissertation has provided a comprehensive review on the research into modelling human motion with a focus on data-driven machine learning approaches, aiming not only to present a summary of many of the major models and approaches to character motion synthesis but also to put that information within the context of how the whole pipeline is developed, starting from human motion data capture stage and ending with the generative block for synthesizing character animation in Chapter 2. The dissertation also covered a number of research directions and presented a number of contributions with the aim of improving character animation pipelines and achieving high quality and more believable virtual characters. These contributions include human motion datasets, data pre-processing tools, generative models for human motion, and noteworthy findings that are believed to be helpful in furthering the state-of-the-art in the field of character animation.

First, in Chapter 3 we presented MoVi, a large synchronized and cross-calibrated human motion and video dataset enriched with dynamic body shape parameters as part of the AMASS project. MoVi was recorded in a large number of different action types, paving the way for our multi-modal human motion modelling in the later projects. In addition, the recorded synchronized and cross-calibrated video is considered a significant contribution to the research field of body shape and pose estimation from video data.

Second, in Chapter 4, we proposed an approach for automating the marker labelling stage in the optical motion capture pipelines that can be used to alleviate the pain of manually labelling the recorded marker trajectories as one of the main friction points in the optical motion capture pipelines. We developed a novel end-to-end data-driven model for initializing and tracking the optical markers, which leverages the power of deep learning models in learning complex mappings. We formulated the marker initialization as a permutation learning problem where the unlabelled markers are sorted by a permutation matrix given their 3D positions. Since the space of permutation matrices is discrete, we proposed the idea of relaxing our objective function by using doubly-stochastic matrices, which can be considered as a continuous approximation of permutation matrices. In addition, we proposed using a differentiable Sinkhorn normalization layer to ensure that the output of our model lies on the manifold of doubly-stochastic matrices. In the next step, we explored different voting-based schemes to correct the remaining labelling errors. Our model

was compact, efficient, and considerably more accurate than the previous methods, especially when marker occlusions occur. To demonstrate the effectiveness of our model, we compared both initialization and error-correction steps with other state-of-the-art methods. Our model performs with high accuracy even for short recorded data or when the optical markers are occluded; however, the number of markers should be fixed, and a separate model should be trained for each specific marker layout.

Third, in Chapter 5, we proposed a hierarchical probabilistic model for generating character animation given weak control signals such as action type or gender. Only requiring weak control signals and probabilistic modelling, our framework can synthesize a diverse set of motion primitives with minimum supervision. This turns our framework into a useful tool for 1) providing synthetic data to be used in other character animation blocks and 2) generating batches of motion primitives for crown simulation with very low latency. The hierarchical probabilistic structure of our model helped address two main challenges simultaneously, modelling the natural variations in human motion and avoiding mean collapse, which usually happens in deterministic models. In the outer layer, we proposed using Motion Words which represent a short sequence of poses. We formulated the inner layer of our architecture using a variational recurrent neural network where the prior and posterior distributions are a function of control signals and previous time steps. Injecting noise at the Motion Word level significantly increased the motion variability and improved the quality of generated motions. In addition, we improved the results by carefully designing the reconstruction loss using the weighted sum of geodesic distances where the importance of each joint is weighted based on its position in the skeleton hierarchy. The quantitative and qualitative evaluation results corroborated the high performance of our model, which can be used for crowd simulation and generating synthetic motion data. However, our model was limited to the number of action types as the quality of generated motion decreased when the number of actions increased. In addition, we did not incorporate any information about the surrounding environment.

Lastly, in Chapter 6, we proposed a probabilistic architecture for character control using normalizing flows. Our architecture incorporated several ideas to be efficient and compatible with the current gaming platforms while producing realistic human motion with natural variations. In this work, we extended and improved MoGlow, the previous flow-based model for character animation, across different directions. We modelled the dynamics not only in the coupling flows but also in

permutation layers and the newly proposed Temporal Affine Layer. In our model, not only the pose but also the character's movement and the predicted future trajectory are modelled probabilistically, providing a more accurate simulation of human motion. Although our model is able to render pose states at 60 fps, it is still slower than the most common approaches in game production, such as motion matching, leaving less room for other computations such as environment interactions in a normal machine. In addition, we did not model the complex environment interactions such as sitting on a chair or lifting a box. Finally, this model is limited to the number of action types similar to the previously proposed model.

7.2 Future Directions

This dissertation outlines several potential research directions for the future. A number of promising directions are discussed here, albeit this is not a comprehensive list.

7.2.1 Towards Motion Capture Data Processing

There has been little effort in developing unified and standard libraries and packages for motion capture data processing, including file readers and parsers, dataset loaders, feature extractors, representation converters, analysis, and visualization tools, even though there has been a lot of research and work in this area. As a result, most researchers must build all of the pipeline's blocks from the ground up. Furthermore, there are no consistent representations of motion and skeleton. All of these issues make comparing multiple models and using diverse datasets in a unified framework problematic.

Regarding marker labelling, despite promising results in using machine learning approaches for automatic labelling of optical markers, they are not yet mature enough to be used in the mocap software. Ghorbani and Black [146] implicitly used prior information about the body shape and improved our results, especially when occlusions occur. However, their model is also based on an assumption of a fixed marker layout and cannot handle additional hand and face markers. One potential direction is to expand such models to handle different marker layouts by using body prior information and finding the closest corresponding vertex to each query marker by optimizing over the space of the pose and body shape parameters. This idea can also be extended for the

multi-subject scenarios.

7.2.2 Towards Motion Modelling and Character Animation

The majority of proposed models in character control and human motion modelling are based on deterministic approaches that ignore human motion's stochasticity and natural variations. Even if such models generate high-quality samples, they still lack a convincing diversity, causing duplicate motion samples that could be noticeable in long-term scenarios such as video games. Generative models have shown a remarkable potential to address this problem. However, they face other challenges such as training instability in GANs, computational complexity in normalizing flows and autoregressive models, and overly smooth samples in VAEs. Nevertheless, these models represent a laudable push toward realistic motion synthesis and are worth being further investigated. Another future direction is to facilitate integrating style into the generated motion. Several works have been proposed to this end [187, 33, 188, 86, 41, 70]. Yet, none of the proposed approaches are mature enough to be used in production. One of the main challenges in this direction is defining and representing the style component expressive enough to reflect motion variations in different dimensions. The model should also be able to extract/integrate such style components from/into the motion. Another challenge in this direction is to generate stylistic movement in real-time, which is a crucial requirement in video games or interactive scenarios. Unlike direct data-driven approaches such as motion graphs and motion matching, machine learning frameworks are limited by the number of generated action types. This issue arises from different limitations such as the mean collapse problem, large variations in each action class, model convergence for very large datasets, and learning transitions between different actions. Partitioning the training data space into multiple regimes and training separate models for each regime in frameworks such as Mixture of Experts [12] and Mode-adaptive Neural network [5] may provide a way for generating samples from different action types. However, the network parameters are in a linear order of covered actions.

Bibliography

[1] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4346–4354, 2015.

[2] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2891–2900, 2017.

[3] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 138, 2016.

[4] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 42, 2017.

[5] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–11, 2018.

[6] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables," in *Advances in neural information processing systems*, pp. 1345–1352, 2007.

[7] G. W. Taylor and G. E. Hinton, "Factored conditional restricted boltzmann machines for modeling motion style," in *Proceedings of the 26th annual international conference on machine learning*, pp. 1025–1032, 2009.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[9] L.-Y. Gui, Y.-X. Wang, X. Liang, and J. M. Moura, "Adversarial geometry-aware human motion prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 786–803, 2018.

[10] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[12] H. Y. Ling, F. Zinno, G. Cheng, and M. Van De Panne, "Character controllers using motion vaes," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 40–1, 2020.

[13] D. Greenwood, S. Laycock, and I. Matthews, "Predicting head pose from speech with a conditional variational autoencoder," ISCA, 2017.

[14] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, pp. 2980–2988, 2015.

[15] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, "Amass: Archive of motion capture as surface shapes," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5442–5451, 2019.

[16] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *ACM SIGGRAPH 2008 classes*, pp. 1–10, 2008.

[17] S. Clavet, "Motion matching and the road to next-gen animation," in *Proc. of GDC*, 2016.

[18] "C-motion research biomechanics." http://www2.c-motion.com/index.php.

[19] M. Loper, N. Mahmood, and M. J. Black, "Mosh: Motion and shape capture from sparse markers," *ACM Transactions on Graphics*, vol. 33, no. 6, p. 220, 2014.

[20] S. Ghorbani, A. Etemad, and N. F. Troje, "Auto-labelling of markers in optical motion capture by permutation learning," in *Computer Graphics International Conference*, pp. 167–178, Springer, 2019.

[21] S. Holzreiter, "Autolabeling 3d tracks using neural networks," *Clinical Biomechanics*, vol. 20, no. 1, pp. 1–8, 2005.

[22] J. Maycock, T. Röhlig, M. Schröder, M. Botsch, and H. Ritter, "Fully Automatic Optical Motion Tracking using an Inverse Kinematics Approach," . *In IEEE/RAS International Conference on Humanoid Robots*, pp. 2–7, 2015.

[23] S. Han, B. Liu, R. Wang, Y. Ye, C. D. Twigg, and K. Kin, "Online optical marker-based hand tracking with deep labels," *ACM Transactions on Graphics*, vol. 37, no. 4, p. 166, 2018.

[24] D. Holden, "Robust Solving of Optical Motion Capture Data by Denoising," *ACM Transactions on Graphics*, vol. 38, no. 1, pp. 1–12, 2018.

[25] S. Ghorbani, C. Wloka, A. Etemad, M. A. Brubaker, and N. F. Troje, "Probabilistic character motion synthesis using a hierarchical deep latent variable model," in *Computer Graphics Forum*, vol. 39, pp. 225–239, Wiley Online Library, 2020.

[26] D. Holden, J. Saito, T. Komura, and T. Joyce, "Learning motion manifolds with convolutional autoencoders," in *SIGGRAPH Asia 2015 Technical Briefs*, pp. 1–4, 2015.

[27] G. E. Henter, S. Alexanderson, and J. Beskow, "Moglow: Probabilistic and controllable motion synthesis using normalising flows," *arXiv preprint arXiv:1905.06598*, 2019.

[28] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[29] D. Pavllo, C. Feichtenhofer, M. Auli, and D. Grangier, "Modeling human motion with quaternion-based neural networks," *International Journal of Computer Vision*, pp. 1–18, 2019.

[30] S. Ghorbani, C. Wloka, A. Etemad, M. A. Brubaker, and N. F. Troje, "Probabilistic Character Motion Synthesis using a Hierarchical Deep Latent Variable Model," *Computer Graphics Forum*, 2020.

[31] S. Starke, H. Zhang, T. Komura, and J. Saito, "Neural state machine for character-scene interactions.," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 209–1, 2019.

[32] S. Starke, Y. Zhao, T. Komura, and K. Zaman, "Local motion phases for learning multi-contact character movements," *ACM Transactions on Graphics*, vol. 39, 06 2020.

[33] S. Alexanderson, G. E. Henter, T. Kucherenko, and J. Beskow, "Style-controllable speech-driven gesture synthesisusing normalising flows," in *EUROGRAPHICS 2020*, 2020.

[34] K. Yamane and Y. Nakamura, "Natural motion animation through constraining and deconstraining at will," *IEEE Transactions on visualization and computer graphics*, vol. 9, no. 3, pp. 352–360, 2003.

[35] "Autodesk MotionBuilder software." https://www.autodesk.com/products/motionbuilder/overview, 2021.

[36] "Blender software." https://www.blender.org/, 2021.

[37] A. Bruderlin and L. Williams, "Motion signal processing," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 97–104, 1995.

[38] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[39] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[40] N. Lawrence, "Gaussian process latent variable models for visualisation of high dimensional data," *Advances in neural information processing systems*, vol. 16, pp. 329–336, 2003.

[41] M. Brand and A. Hertzmann, "Style machines," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 183–192, 2000.

[42] G. Liu, M. Xu, Z. Pan, and A. E. Rhalibi, "Human motion generation with multifactor models," *Computer Animation and Virtual Worlds*, vol. 22, no. 4, pp. 351–359, 2011.

[43] D. Pavllo, D. Grangier, and M. Auli, "Quaternet: A quaternion-based recurrent model for human motion," *arXiv preprint arXiv:1805.06485*, 2018.

[44] S. Starke, H. Zhang, T. Komura, and J. Saito, "Neural state machine for character-scene interactions," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–14, 2019.

[45] R. Gross and J. Shi, "The cmu motion of body (mobo) database," 2001.

[46] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, vol. 34, pp. 248:1–248:16, Oct. 2015.

[47] M. Buttner, "Machine learning for motion synthesis and character control in games," *Proc. of I3D 2019*, 2019.

[48] N. F. Troje, "Retrieving information from human movement patterns," *Understanding events: How humans see, represent, and act on events*, vol. 1, pp. 308–334, 2008.

[49] O. Alemi and P. Pasquier, "Machine learning for data-driven movement generation: a review of the state of the art," *arXiv preprint arXiv:1903.08356*, 2019.

[50] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *ACM SIGGRAPH 2008 classes*, pp. 1–10, 2008.

[51] R. Bowden, "Learning statistical models of human motion," in *IEEE Workshop on Human Modeling, Analysis and Synthesis, CVPR*, vol. 2000, 2000.

[52] J. Chai and J. K. Hodgins, "Performance animation from low-dimensional control signals," in *ACM SIGGRAPH 2005 Papers*, pp. 686–696, 2005.

[53] J. Chai and J. K. Hodgins, "Constraint-based motion optimization using a statistical dynamic model," in *ACM SIGGRAPH 2007 papers*, pp. 8–es, 2007.

[54] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[55] J. Tilmanne and T. Dutoit, "Expressive gait synthesis using pca and gaussian modeling," in *International Conference on Motion in Games*, pp. 363–374, Springer, 2010.

[56] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pp. 245–254, 1985.

[57] N. F. Troje, "Decomposing biological motion: A framework for analysis and synthesis of human gait patterns," *Journal of vision*, vol. 2, no. 5, pp. 2–2, 2002.

[58] H. Qu, Z. Yu, X. Wang, and H.-S. Wong, "Motion synthesis based on dimensionality reduction," in *2008 First IEEE International Conference on Ubi-Media Computing*, pp. 237–242, IEEE, 2008.

[59] A.-A. Samadani, E. Kubica, R. Gorbet, and D. Kulić, "Perception and generation of affective hand movements," *International Journal of Social Robotics*, vol. 5, no. 1, pp. 35–51, 2013.

[60] J. O. Ramsay, "Functional data analysis," *Encyclopedia of Statistical Sciences*, vol. 4, 2004.

[61] J. Min and J. Chai, "Motion graphs++ a compact generative model for semantic motion analysis and synthesis," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, pp. 1–12, 2012.

[62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[63] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[64] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, PMLR, 2015.

[65] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[66] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[67] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proceedings of the ieee conference on computer vision and pattern recognition*, pp. 5308–5317, 2016.

[68] E. Aksan, M. Kaufmann, and O. Hilliges, "Structured prediction helps 3d human motion modelling," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7144–7153, 2019.

[69] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.," *Journal of machine learning research*, vol. 11, no. 12, 2010.

[70] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," in *ACM SIGGRAPH 2004 Papers*, pp. 522–531, 2004.

[71] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–10, 2012.

[72] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2414–2423, 2016.

[73] K. Aberman, Y. Weng, D. Lischinski, D. Cohen-Or, and B. Chen, "Unpaired motion style transfer from video to animation," *arXiv preprint arXiv:2005.05751*, 2020.

[74] H. Y. Ling, F. Zinno, G. Cheng, and M. van de Panne, "Character controllers using motion vaes," vol. 39, no. 4, 2020.

[75] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[76] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[77] N. Lawrence and A. Hyvärinen, "Probabilistic non-linear principal component analysis with gaussian process latent variable models.," *Journal of machine learning research*, vol. 6, no. 11, 2005.

[78] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 283–298, 2007.

[79] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[80] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," tech. rep., Colorado Univ at Boulder Dept of Computer Science, 1986.

[81] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[82] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bulletin of the American Mathematical Society*, vol. 73, no. 3, pp. 360–363, 1967.

[83] L. E. Baum and G. Sell, "Growth transformations for functions on manifolds," *Pacific Journal of Mathematics*, vol. 27, no. 2, pp. 211–227, 1968.

[84] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[85] P. Baldi and Y. Chauvin, "Smooth on-line learning algorithms for hidden markov models," *Neural Computation*, vol. 6, no. 2, pp. 307–318, 1994.

[86] Y. Wang, Z.-Q. Liu, and L.-Z. Zhou, "Learning style-directed dynamics of human motion for automatic motion synthesis," in *2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 4428–4433, IEEE, 2006.

[87] D. Herzog and V. Krüger, "Recognition and synthesis of human movements by parametric hmms," in *Statistical and Geometrical Approaches to Visual Motion Analysis*, pp. 148–168, Springer, 2009.

[88] D. Herzog, V. Krüger, and D. Grest, "Parametric hidden markov models for recognition and synthesis of movements.," in *BMVC*, vol. 8, pp. 163–172, Citeseer, 2008.

[89] E. Barsoum, J. Kender, and Z. Liu, "Hp-gan: Probabilistic 3d human motion prediction via gan," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 1418–1427, 2018.

[90] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[91] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[92] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.

[93] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "Made: Masked autoencoder for distribution estimation," in *International Conference on Machine Learning*, pp. 881–889, 2015.

[94] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in neural information processing systems*, pp. 4790–4798, 2016.

[95] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[96] S. Reed, A. v. d. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. De Freitas, "Parallel multiscale autoregressive density estimation," *arXiv preprint arXiv:1703.03664*, 2017.

[97] D. Weissenborn, O. Täckström, and J. Uszkoreit, "Scaling autoregressive video models," *arXiv preprint arXiv:1906.02634*, 2019.

[98] P. Ramachandran, T. L. Paine, P. Khorrami, M. Babaeizadeh, S. Chang, Y. Zhang, M. A. Hasegawa-Johnson, R. H. Campbell, and T. S. Huang, "Fast generation for convolutional autoregressive models," *arXiv preprint arXiv:1704.06001*, 2017.

[99] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *arXiv preprint arXiv:1906.02691*, 2019.

[100] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, pp. 3483–3491, 2015.

[101] I. Habibie, D. Holden, J. Schwarz, J. Yearsley, T. Komura, J. Saito, I. Kusajima, X. Zhao, M.-G. Choi, R. Hu, *et al.*, "A recurrent variational autoencoder for human motion synthesis.," in *BMVC*, 2017.

[102] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[103] Y. Rubanova, R. T. Chen, and D. Duvenaud, "Latent odes for irregularly-sampled time series," *arXiv preprint arXiv:1907.03907*, 2019.

[104] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *European Conference on Computer Vision*, pp. 835–851, Springer, 2016.

[105] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*, pp. 1530–1538, PMLR, 2015.

[106] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *arXiv preprint arXiv:1908.09257*, 2019.

[107] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[108] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[109] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[110] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

[111] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A flow-based generative network for speech synthesis," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3617–3621, IEEE, 2019.

[112] S. Kim, S.-g. Lee, J. Song, J. Kim, and S. Yoon, "Flowavenet: A generative flow for raw audio," *arXiv preprint arXiv:1811.02155*, 2018.

[113] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma, "Videoflow: A flow-based generative model for video," *arXiv preprint arXiv:1903.01434*, vol. 2, no. 5, 2019.

[114] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 483–490, 2002.

[115] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *arXiv preprint arXiv:1606.03498*, 2016.

[116] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *arXiv preprint arXiv:1706.08500*, 2017.

[117] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[118] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn, "Robonet: Large-scale multi-robot learning," *arXiv preprint arXiv:1910.11215*, 2019.

[119] L. Marco and G. M. Farinella, *Computer Vision for Assistive Healthcare*. Academic Press, 2018.

[120] N. Hesse, S. Pujades, M. Black, M. Arens, U. Hofmann, and S. Schroeder, "Learning and tracking the 3d body shape of freely moving infants from rgb-d sequences," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[121] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "Smpl: A skinned multi-person linear model," *ACM transactions on graphics (TOG)*, vol. 34, no. 6, p. 248, 2015.

[122] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, "Amass: Archive of motion capture as surface shapes," *arXiv preprint arXiv:1904.03278*, 2019.

[123] T. von Marcard, B. Rosenhahn, M. J. Black, and G. Pons-Moll, "Sparse inertial poser: Automatic 3d human pose estimation from sparse imus," in *Computer Graphics Forum*, vol. 36, pp. 349–360, Wiley Online Library, 2017.

[124] L. Sigal, A. O. Balan, and M. J. Black, "Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion," *International journal of computer vision*, vol. 87, no. 1-2, p. 4, 2010.

[125] C. Mandery, Ö. Terlemez, M. Do, N. Vahrenkamp, and T. Asfour, "The kit whole-body human motion database," in *2015 International Conference on Advanced Robotics (ICAR)*, pp. 329–336, IEEE, 2015.

[126] F. De la Torre, J. Hodgins, J. Montano, S. Valcarcel, R. Forcada, and J. Macey, "Guide to the carnegie mellon university multimodal activity (cmu-mmac) database," *Robotics Institute, Carnegie Mellon University*, vol. 5, 2009.

[127] M. Trumble, A. Gilbert, C. Malleson, A. Hilton, and J. Collomosse, "Total capture: 3d human pose estimation fusing video and inertial sensors.," in *BMVC*, vol. 2, p. 3, 2017.

[128] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013.

[129] M. Loper, N. Mahmood, and M. J. Black, "Mosh: Motion and shape capture from sparse markers," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, p. 220, 2014.

[130] A. Qualisys, "Qualisys track manager user manual," *Gothenburg: Qualisys AB*, 2006.

[131] J. Heikkila, O. Silven, *et al.*, "A four-step camera calibration procedure with implicit image correction," in *cvpr*, vol. 97, p. 1106, Citeseer, 1997.

[132] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000.

[133] J. Bouguet, "Camera calibration toolbox for matlab." http://www.vision.caltech.edu/bouguetj/calib_doc/. Accessed: 2019-11-07.

[134] "Noitom. axis neuron user manual.." https://neuronmocap.com/system/files/software/Axis%20Neuron%20User%20Manual_V3.8.1.5.pdf.

[135] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

[136] P. V. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962. US Patent 3,069,654.

[137] "Coda Pelvis visual3d wiki documentation." https://c-motion.com/v3dwiki/index.php?title=Coda_Pelvis. Accessed: 2019-11-07.

[138] A. L. Bell, R. A. Brand, and D. R. Pedersen, "Prediction of hip joint centre location from external landmarks," *Human movement science*, vol. 8, no. 1, pp. 3–16, 1989.

[139] A. L. Bell, D. R. Pedersen, and R. A. Brand, "A comparison of the accuracy of several hip center location prediction methods," *Journal of biomechanics*, vol. 23, no. 6, pp. 617–621, 1990.

[140] "Golem/Plug-in Gait Upper Extremity Model visual3d wiki documentation.." https://www.c-motion.com/v3dwiki/index.php?title=Tutorial:_Plug-In_Gait_Full-Body.

[141] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "Scape: shape completion and animation of people," in *ACM SIGGRAPH 2005 Papers*, pp. 408–416, 2005.

[142] S. Ghorbani, K. Mahdaviani, A. Thaler, K. Kording, D. J. Cook, G. Blohm, and N. F. Troje, "MoVi: A Large Multipurpose Motion and Video Dataset," 2020.

[143] O. Matthews, K. Ryu, and T. Srivastava, "Creating a large-scale synthetic dataset for human activity recognition," *arXiv preprint arXiv:2007.11118*, 2020.

[144] L. Herda, P. Fua, R. Plänkers, R. Boulic, and D. Thalmann, "Using skeleton-based tracking to increase the reliability of optical motion capture," *Human Movement Science*, vol. 20, no. 3, pp. 313–341, 2001.

[145] Q. Yu, Q. Li, and Z. Deng, "Online motion capture marker labeling for multiple interacting articulated targets," *Computer Graphics Forum*, vol. 26, no. 3, pp. 477–483, 2007.

[146] N. Ghorbani and M. J. Black, "Soma: Solving optical marker-based mocap automatically," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11117–11126, 2021.

[147] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[148] G. Peyré, M. Cuturi, *et al.*, "Computational optimal transport: With applications to data science," *Foundations and Trends® in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[149] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. Osman, D. Tzionas, and M. J. Black, "Expressive body capture: 3d hands, face, and body from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10975–10985, 2019.

[150] J. Meyer, M. Kuderer, J. Muller, and W. Burgard, "Online marker labeling for fully automatic skeleton Tracking in optical motion capture," *In Proc. IEEE International Conference on Robotics and Automation*, pp. 5652–5657, 2014.

[151] T. Schubert, A. Gkogkidis, T. Ball, and W. Burgard, "Automatic initialization for skeleton tracking in optical motion capture," in *Proc. IEEE International Conference on Robotics and Automation*, pp. 734–739, IEEE, 2015.

[152] R. P. Adams and R. S. Zemel, "Ranking via Sinkhorn Propagation," *ArXiv*, pp. 1106–1925, 2011.

[153] R. Santa Cruz, B. Fernando, A. Cherian, and S. Gould, "Deeppermnet: Visual permutation learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3949–3957, 2017.

[154] S. H. Rezatofighi, R. Kaskman, F. T. Motlagh, Q. Shi, D. Cremers, L. Leal-Taixé, and I. Reid, "Deep perm-set net: learn to predict sets with unknown permutation and cardinality using deep neural networks," *arXiv preprint arXiv:1805.00613*, 2018.

[155] G. Birkhoff, "Three observations on linear algebra," *Univ. Nac. Tacuman, Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[156] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[157] R. Sinkhorn, "A relationship between arbitrary positive matrices and doubly stochastic matrices," *The Annals of Mathematical Statistics*, vol. 35, no. 2, pp. 876–879, 1964.

[158] P. A. Knight, "The sinkhorn–knopp algorithm: convergence and applications," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 261–275, 2008.

[159] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[160] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[161] S. Ghorbani, K. Mahdaviani, A. Thaler, K. Kording, D. J. Cook, G. Blohm, and N. F. Troje, "Movi: A large multipurpose motion and video dataset," *arXiv preprint arXiv:2003.01888*, 2020.

[162] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[163] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, pp. 2234–2242, 2016.

[164] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, pp. 6626–6637, 2017.

[165] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–14, 2018.

[166] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *arXiv preprint arXiv:1511.06349*, 2015.

[167] D. Q. Huynh, "Metrics for 3d rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 2009.

[168] L.-Y. Gui, Y.-X. Wang, X. Liang, and J. M. Moura, "Adversarial geometry-aware human motion prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 786–803, 2018.

[169] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables," in *Advances in neural information processing systems*, pp. 1345–1352, 2007.

[170] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[171] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[172] P. Ghosh, J. Song, E. Aksan, and O. Hilliges, "Learning human motion models for long-term predictions," in *2017 International Conference on 3D Vision (3DV)*, pp. 458–466, IEEE, 2017.

[173] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[174] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *arXiv preprint arXiv:1912.02762*, 2019.

[175] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[176] M. Büttner and S. Clavet, "Motion matching - the road to next gen animation," 2015.

[177] E. Barsoum, J. Kender, and Z. Liu, "Hp-gan: Probabilistic 3d human motion prediction via gan," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1418–1427, 2018.

[178] Z. Wang, J. Chai, and S. Xia, "Combining recurrent neural networks and adversarial training for human motion synthesis and control," *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[179] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[180] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[181] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," *Advances in neural information processing systems*, vol. 29, pp. 4743–4751, 2016.

[182] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," *arXiv preprint arXiv:1705.07057*, 2017.

[183] D. Holden, O. Kanoun, M. Perepichka, and T. Popa, "Learned motion matching," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 53–1, 2020.

[184] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *International Conference on Machine Learning*, pp. 2722–2730, PMLR, 2019.

[185] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[186] W. Falcon, "Pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning Cited by*, vol. 3, 2019.

[187] K. Aberman, Y. Weng, D. Lischinski, D. Cohen-Or, and B. Chen, "Unpaired motion style transfer from video to animation," *arXiv preprint arXiv:2005.05751*, 2020.

[188] H. J. Smith, C. Cao, M. Neff, and Y. Wang, "Efficient neural networks for real-time motion style transfer," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 2, no. 2, pp. 1–17, 2019.