Thesis Dissertation

CROSS-DOMAIN MOTION RETARGETING

Alexios Mylordos

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

MAY 2024

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Cross-Domain Motion Retargeting

Alexios Mylordos

Supervisor

Dr. Andreas Aristidou

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor's in Computer Science at University of Cyprus

May 2024

Acknowledgements

I express my deepest gratitude to my supervisor Dr. Andreas Aristidou for his unwavering guidance and support throughout the duration of this thesis. His expertise, patience, and valuable insights have been pivotal in steering the course and ensuring the success of this research journey.

Additionally, I wish to extend my heartfelt appreciation to my friends and family who stood by me with steadfast encouragement, and patience. Your collective support has been instrumental in the successful completion of this endeavor, and I am profoundly grateful for your invaluable contributions.

Finally, I would like to sincerely thank Jose Louis Ponton for his willingness and readiness to provide assistance and insightful feedback.

Contents

Abstract

An increasing number of traditional problems in the field of computer graphics are successfully being tackled with data-driven machine learning approaches, and as a result, the demand for high quality ample data is becoming more pronounced. Computer animation applications in particular, call for extensive motion capture data from a wide variety of skeletal structures, yet in many cases such as in collection of motion capture data from animals, gathering said data might be challenging or even impossible. To address this problem, we present a novel Cross-Domain Motion Retargeting framework, by which the human skeleton can be effectively used as a motion driver to generate sequences of natural appearing motions in other domains. To achieve this, we put forward two methods that expand the capabilities of a motion reconstruction autoencoder network to facilitate motion retargeting. For the first method, we introduce a corresponding-motion aware training paradigm for the alignment of latent representations of arbitrary domains to that of the human. For the second method, we posit an independent external network which learns mapping functions from arbitrary latent representations to the human latent space, facilitating cross domain retargeting by using the human space as a mediator. In this dissertation we showcase our results that provide evidence for the efficacy of the second method, and although our results are limited, we are optimistic about the potential of our approach and its capacity to illuminate a pathway towards addressing the challenges of animal motion capture data collection.

Chapter 1

Introduction

1.1 The Problem of Motion Retargeting

One of the longest-standing and pivotal problems in Computer Graphics is that of motion retargeting – the act of transferring motion from one skeleton to another, ideally adapting to the target's traits, gaits, and general characteristics. Mocap (motion capture) data finds use in a diverse range of areas such as biomedical research, film, virtual reality, video games, sports performance analysis, and many more. In the particular case of animal mocap data the issue of scarcity prevails despite advancements in the field of computer animation, thus in recent years, various approaches are being explored with the goal of generating high-quality novel animal motions. Motion Retargeting is one such approach. The issue this approach is facing however, is mainly rooted in the inherent difficulty of transferring motion between skeletons with significantly different topologies and traits. It is non-trivial to retarget motions between say, a biped and a quadruped, while maintaining a realistic and believable appearance in the retargeted result. Domain discrepancies, geometric misalignment, and differences in articulation variability make the problem of transferring motions between domains

particularly challenging. Thus, overcoming these obstacles and achieving Cross-Domain Motion Retargeting, is a problem which demands vigorous effort and sophisticated methods, and remains essential for the betterment of the aforementioned areas of application.

1.2 Motivation

Every day, new machine learning models are successfully leveraged to address a variety of problems across a diverse range of fields. The innate versatility of neural networks is what allows us to use machine learning as a seemingly universal solution to countless problems, with the primary substance fueling this development being training data. Adequate and high-quality training data is the fundamental determinant of model efficacy. As humanity continues its digital transformation, propelled by advancements in artificial intelligence, the importance of training data abundance is increasingly emphasized.

In certain scenarios, accumulating sufficient data might pose a challenge. Such is the case of animal motion capture, in which the logistics involved in capturing animal movement, the unpredictability inherent in animal behavior, and the ethical considerations, render the task of acquiring extensive motion capture data from non-human subjects difficult at best, and in some cases impossible.

Combining the urgent need of ample training data with the innate hurdles of acquiring motion capture data from animals, we arrive at the central aim of this thesis. We introduce a novel scalable framework for effectively translating motions from one domain to corresponding and natural looking motions of another domain, regardless of skeletal structure dissimilarity. Our goal is to foster an all-to-all connectivity between domains by aligning encoded representations of motions of animals and humans, effectively acquiring the ability to generate animal motions by use of the human body as a sort of puppeteer (motion driver). By merging a framework as described above with a system that reliably reconstructs motions

from sparse inputs, we realize a system which makes use of readily available equipment such as VR controllers, for the real-time generation of genuine-in-appearance animal motions.

1.3 Scope & Contribution

The scope of this thesis is to first assemble a robust base system for facilitating accurate motion reconstruction of arbitrary skeletal structures using a deep-learning method, along with a motion synthesis mechanism for the purpose of training-data augmentation. Secondly, we aim to describe a scalable framework which augments our base system so as to allow for Cross-Domain Motion Retargeting.

Our main contribution includes the proposal of two methods for achieving motion retargeting. Our first method, “Shared Latent Space” is based on a training paradigm designed to align latent spaces of arbitrary domains to that of the human domain – along with providing the possibility of maintaining either the stride phase or root velocity from the source domain. Our second method “Latent Space Transfer” implements a small external neural network which learns to map one latent space representation onto another by training on pairs of corresponding motion sequences in two domains.

Chapter 2

Related work

2.1 Motion Retargeting

2.1.1 Overview

Several recent works attempt to retain a natural appearance in retargeted motions by employing consistency loss based methods [1][2], but they either suffer from transferring traits of the source motion to the target, or they do not accommodate retargeting for structurally dissimilar skeletons. Another recent work utilizes a physics-based approach [3], achieving real-time generation of plausible motions from a sparse set of inputs, however they do not delve outside the realm of bipedal skeletons. Our aim in this thesis is to introduce a framework which successfully addresses these challenges and provides a reliable and scalable system which can be used for novel motion synthesis in a diverse set of domains, driven by human motions.

2.1.2 Physics-based Motion Retargeting from Sparse Inputs

Reda et al. [3] employ a deep reinforcement learning model, which learns to generate torques for specific points of the skeleton, based on a policy which rewards the network for:

1. Accurate motion generations (by comparing to motion capture data)
2. Simultaneity between foot contact of simulated character and mocap data
3. Low total amount of energy consumed by character

They achieve retargeting by first performing a rough kinematic retargeting from human motion to simulated character motion, and then pass the retargeted motions through the physics simulator which is trained by adhering to the previously explained policy.

The system excels in Physics-based simulations, providing physically plausible animations for the retargeted motions. It performs real-time retargeting of user inputs, using a head mounted display & left/right hand controllers. Due to the lack of lower body inputs, the system struggles to generate accurate results for motions in which the movement of the top and bottom halves of the body are not related in some way. Additionally, Reda et al. remain within the realm of bipedal skeletons, acknowledging the increased challenge of retargeting to a more morphologically dissimilar skeleton e.g. quadrupeds.

2.1.3 Pose-to-Motion: Cross-Domain Motion Retargeting with Pose Prior

In their paper, Q. Zhao et al. [2] propose a novel approach for motion retargeting in the absence of extensive data. Their system leverages an asymmetric CycleGAN, which learns to generate plausible sets of poses and root transformations in the target domain T, from a given motion sequence in the source domain S. Initially a generator translates motion data from domain S to T, and a pose discriminator compares the generated poses to real static pose data from the target domain T, and provides feedback to the generator. Then, another generator-discriminator set maps the generated poses back to the source domain S, ensuring that the generated motions adhere to the original motion characteristics.

The approach is robust in handling different characters with significantly different topologies (dog, bipedal dinosaur, horse, hamster) even with small pose datasets, and generates high-quality motion sequences that are both plausible and diverse. As a result of the limited motion

data in the target domain however, the generated motions might contain motion traits from the source domain that are unrealistic or physically infeasible for the target domain.

2.1.4 Neural Kinematic Networks for Unsupervised Motion Retargetting

The authors of this work [1] successfully demonstrate the capability to transfer motion data between different skeletal structures without using paired motion data, by taking advantage of topological similarities between the skeletons. They propose a neural kinematic network with an adversarial cycle consistency training objective: a cyclic consistency loss is employed to ensure that retargeted motions generate the original motion when retargeted back. The generated motion must appear realistic given the target domain's characteristics – this is addressed by an adversarially trained discriminator. The architecture itself relies on a Forward Kinematics (FK) layer and two Recurrent Neural Networks that work together to extract motion features from input data, and decode the joint rotations of the target domain from the extracted features. The FK layer is fed the target skeleton's structure along with decoupled rotational information extracted by the RNN network, and generates the retargeted result. To address the challenges of collection paired motion data, they propose a novel training paradigm in which their neural kinematic network is given an input motion from skeleton A, the structure of a skeleton B, and retargets the motion to B. The retargeted motion is re-retargeted back to A, and by employing an adversarial cycle consistency training objective they are able to train their network to facilitate natural appearing generated motions. This work is limited in that the network performs retargeting on a fixed number of joints, as constrained by their retargeting algorithm which is expected to select end-effectors of interest when transferring motions. Additionally, their paper does not delve into motion retargeting across structurally dissimilar domains.

2.2 Framework Foundation

2.2.1 Overview:

In this section, we describe the fundamental building blocks of our framework, which include: Skeleton Aware operations, as described in [6] – for the purpose of building a network able to extract meaningful features from skeletal structures performing motions, a Skeleton-Aware autoencoder network, as described in [5] – which achieves accurate motion reconstruction, and finally a generative model [4], which learns to generate novel motions from a single training example – for the purpose of training data augmentation.

2.2.2 GANimator: Neural Motion Synthesis from a Single Sequence

To address dataset size limitations for our own system, we harness the power of GANimator [4], a generative model that learns to synthesize novel motions from a single short motion sequence.

Li et al. use a multiscale hierarchical generative neural network, where each level is responsible for learning the distribution of temporal patches at a different resolution. What this means is that each level in the network hierarchy is responsible for generating motions at increasingly finer temporal granularities, with the first level generating motions from random noise at a coarse temporal resolution, and the final level generating motions at the finest temporal resolution. The network hierarchy is composed of GANs, one in each level - the GANs in all levels except the first, receive the output of the previous level as input. The high-res motion in each level is fed into a patch-style discriminator, which decides if patches within the generated motion are fake or real (instead of the entire motion).

This novel design is what enables GANimator to synthesize high-quality motions that closely resemble the core elements from the original motions, from a single input motion sequence.

As we will explain in chapter 4, our system heavily relies on the existence of corresponding motions between domains, in order to achieve motion retargeting. Thus, the value of a generative model such as GANimator becomes clear.

2.2.3 SparsePoser: Real-time Full-body Motion Reconstruction from Sparse Data

The contribution of this thesis is a framework which builds off of the SparsePoser [5] generator network, initially designed for the synthesis of human motion only. With our contribution, the network now encompasses the reconstruction of motions from animals, as well as the novel functionality of motion retargeting between different skeletons with substantially dissimilar structures.

SparsePoser is a state of the art full-body motion reconstruction system, which utilizes a novel data-driven method to synthesize high quality full-body human animations from a sparse set of sensors, placed on the pelvis and the five endpoints of the human skeleton (head, hands, feet).

Ponton et al. propose a network which successfully learns the human motion manifold from motion capture data. Their system is able to accurately reconstruct smooth full-body motions, given the sensor input of just 6 devices. This is thanks to the design of their generator network, which is a convolutional-based autoencoder, using skeleton-aware operations [6]. The encoder part of the network is split into two units: a static encoder se and a dynamic encoder de, where se is responsible for learning the static features of a skeleton (offsets) and de learns the dynamic features of a motion (displacements and rotations of 6 sparse joints). A decoder d then takes the encoded motion as input, and reconstructs the original animation. The static features from the static encoder se are then added to the convolution result to consider the static structure of the skeleton. Finally, Ponton et al. employ Inverse Kinematics networks for the arms and legs, for the purpose of learning a more precise positioning of joints, for certain use cases such as VR that might warrant exact placement of end-effectors. We won't however be using it in our framework, sticking with just the generator instead.

Additionally, SparsePoser supports Real-Time Virtual Reality control, by which using only a sparse set of inputs - a head mounted display, two hand-held controllers and three trackers placed on the feet and back – users can animate a full body avatar.

SparsePoser generates animations whose pose quality outperforms the then state-of-the-art methods, and its real time performance capabilities make it suitable for low latency critical applications such as VR. Its main limitation is the reliance on the quality of the training dataset – that is to say, the network may have difficulty generalizing to sparse input that it has not been sufficiently exposed to.

2.2.4 Skeleton-Aware Networks for Deep Motion Retargeting

In order to craft a network that effectively reconstructs motion, a way to extract meaningful features from said motions is required. Aberman et al. [6] propose a framework which incorporates “Skeleton-Aware” operations, such as convolutions and poolings, which allow for precisely this meaningful feature extraction that we need. In the same paper, Aberman et al. present an encoder-decoder network which successfully retargets motion from the domain of one humanoid skeleton to another of a somewhat dissimilar structure, for instance 2 skeletons with the same amount of end effectors, but different bone lengths. To achieve this, they leverage their Skeleton-Aware operations, to reduce poses to a “primal skeleton” which the different skeletons share. By encoding motions into this shared space, the encoder-decoder network facilitates motion retargeting between the different domains, effectively taking advantage of the topological similarities between the skeletons.

Chapter 3

Background

3.1 Autoencoder Networks

In our approach for confronting the daunting task of motion retargeting across different domains, we leverage autoencoder networks [7][8], a type of neural network designed to encode unlabeled input data into some reduced/compressed form, and then decode the compressed data back to the input's original form. The encoded form of the input data is referred to as the latent space.

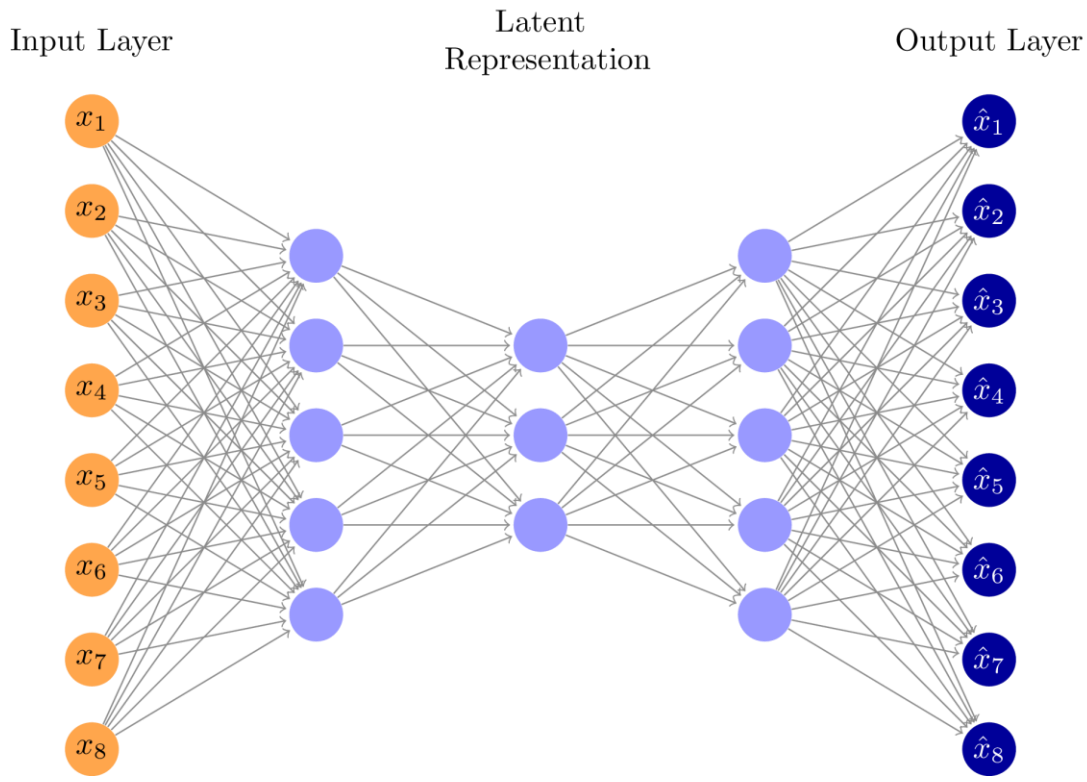

Figure 1: Example of an autoencoder network. The input layer and first hidden layer comprise the encoder, the last two layers the decoder, and the middle layer is called the latent space. Source: https://tikz.net/autoencoder/

Autoencoders excel at preserving important features while reducing the dimensionality of data, and reconstructing the original input from low dimensionality data. As a result, they find applications in a wide range of areas including anomaly detection in cyber security/fraud detection, image denoising, image generation (for variational autoencoder) and many more. In our case, we utilize the strengths of the autoencoder network to achieve accurate motion reconstruction, retargeting and generation (from sparse data).

The three main components of an autoencoder network are the encoder, latent space, and decoder. The input data is first passed through the encoder, which is composed of layers that get progressively smaller, so as to “squeeze” the input into some reduced form. The decoder then uses the lower-dimension data which the encoder outputs, to reconstruct the data back to how it was before it was passed through the encoder.

3.1.1 Encoder

The encoder is typically a feedforward network with the architecture of a multilayer perceptron, or that of a convolutional neural network, depending on the requirements of the application.

3.1.2 Latent Space

The latent space holds the most compressed form of the input. It is the layer that “sits” between the encoder and decoder. In a way, it’s the encoder’s output and the decoder’s input. The fundamental goal of an autoencoder network is to learn an encoding function that produces effective latent space representations - that is to say, the encoder will have discovered the minimum number of important features required for the reconstruction of the input data.

3.1.3 Decoder

The decoder, similar to the encoder, is a feedforward network which contains progressively larger layers, with the first layer having the dimensions of the latent space, and the final layer (which is also the output layer of the entire autoencoder network) having the dimensions of the original input data. The goal of the decoder is to learn a decoding function which will reconstruct the original input data, from the latent space representation.

3.1.4 Training & Loss

During training, the encoder and decoder are trained simultaneously, so that the autoencoder can learn an effective latent representation of the data, and at the same time cultivate the ability to reconstruct it accurately. To achieve this, the network must be trained on a loss function which encourages this behavior, typically a mean squared error (MSE) loss:

$$d(x, x') = \|x - x'\|_2^2$$

Where d is the reconstruction/loss function, x is the original input, x' is the reconstructed data, and $\| \|_2^2$ is the square of the Euclidean norm. Then the problem of optimizing an autoencoder on a dataset $\{x_1, \dots, x_n\}$ becomes a least-squares optimization:

$$\arg\min(L), \qquad L = \frac{1}{N}\sum_{i=1}^{N} \|x_i - D(E(x_i))\|_2^2$$

Where D is the decoding function, and E is the encoding function.

3.2 Pose Representation

3.2.1 Dual Quaternions

We represent poses as dual quaternions as presented by Andreou et al, 2022 [9]. A "regular" quaternion is a mathematical construct which serves as an extension to the complex numbers. A quaternion is defined by the general form:

$$a + bi + cj + dk$$

where *a,b,c,d* are real numbers, and *i,j,k* are symbols that can be interpreted as unit-vectors pointing along the three spatial axes *(x,y,z)*. Quaternions are useful in the field of computer graphics, as they are used as a representation of 3D rotations, thanks to certain advantages they offer against traditional representations. In particular, quaternions are more compact than matrices and quicker to compute, unlike Euler angles they do not suffer from gimbal lock (losing a degree of freedom due to alignment of axes), and they do not cause the "candy wrapper effect", where a loss of volume is observed when interpolating between orientations.

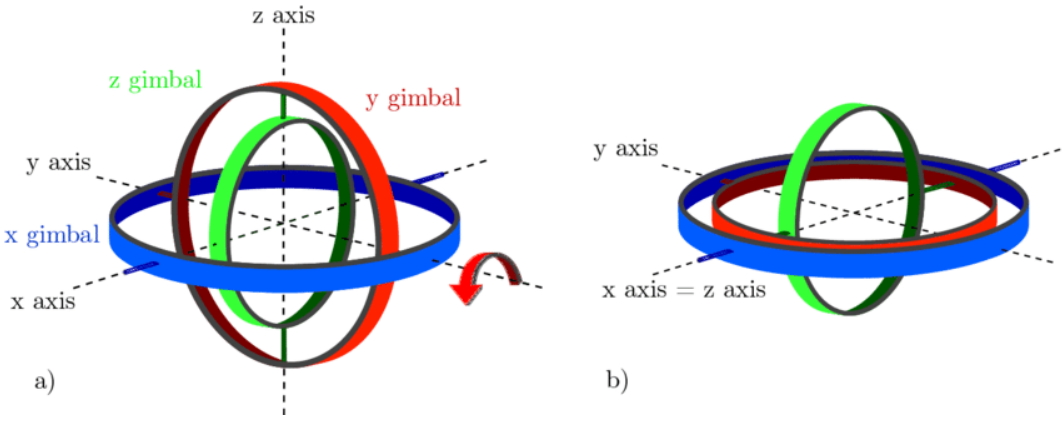

Figure 2: Visual explanation of gimbal lock. In image (b) the x and y gimbals are aligned, thus both of them rotate on the same plane. Source: https://www.researchgate.net/figure/llustrates-the-principle-of-gimbal-lock-The-outer-blue-frame-represents-the-x-axis-

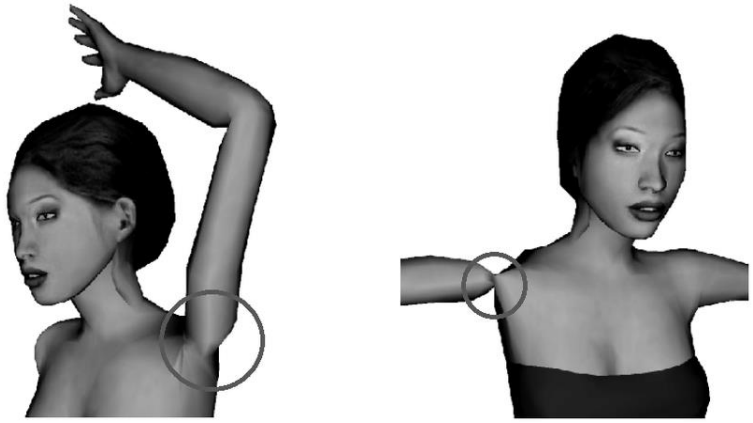

Figure 3: Visual representation of candy-wrapper effect.

Intuitively, we can understand quaternions as representations of rotations in the following way:

Let $q = \cos\left(\frac{\theta}{2}\right) + \vec{u}\sin\left(\frac{\theta}{2}\right)$ be a quaternion that represents a rotation θ about the unit vector $\vec{u}$, where $\vec{u} = bi + cj + dk$, and since $\vec{u}$ is a unit vector, the sum of the squares of the components *b,c,d* is 1.

$$b^2 + c^2 + d^2 = 1$$

Thus, if we imagine *i,j,k* as corresponding to the spatial axes *(x,y,z)*, then for $b = 1$ (and subsequently $c = d = 0$), we have a quaternion q representing the rotation of θ degrees about the x axis,

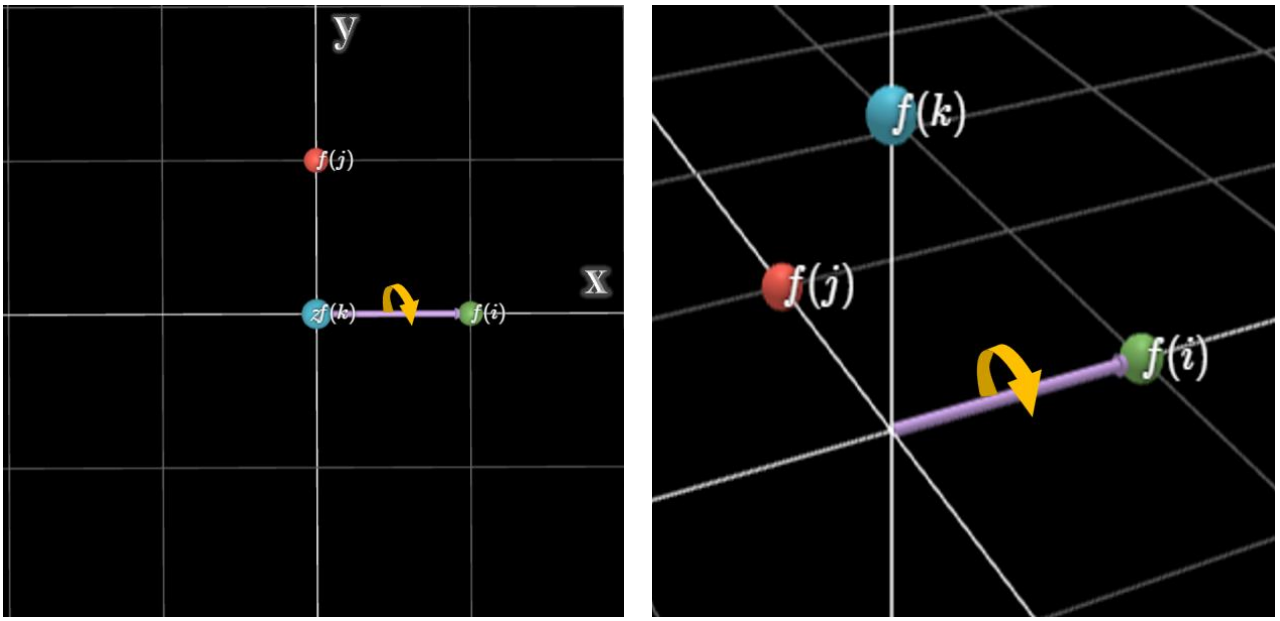

Figure 4: Rotation axis determined by quaternion with b = 1, c = d =0. Rotation direction is given by right hand rule. Source: https://eater.net/quaternions

and for $b \cong 0.71 , c \cong 0.70$ we would have a quaternion representing the rotation of θ degrees around the vector which sits in around the middle of the x,y axes on the xy plane.

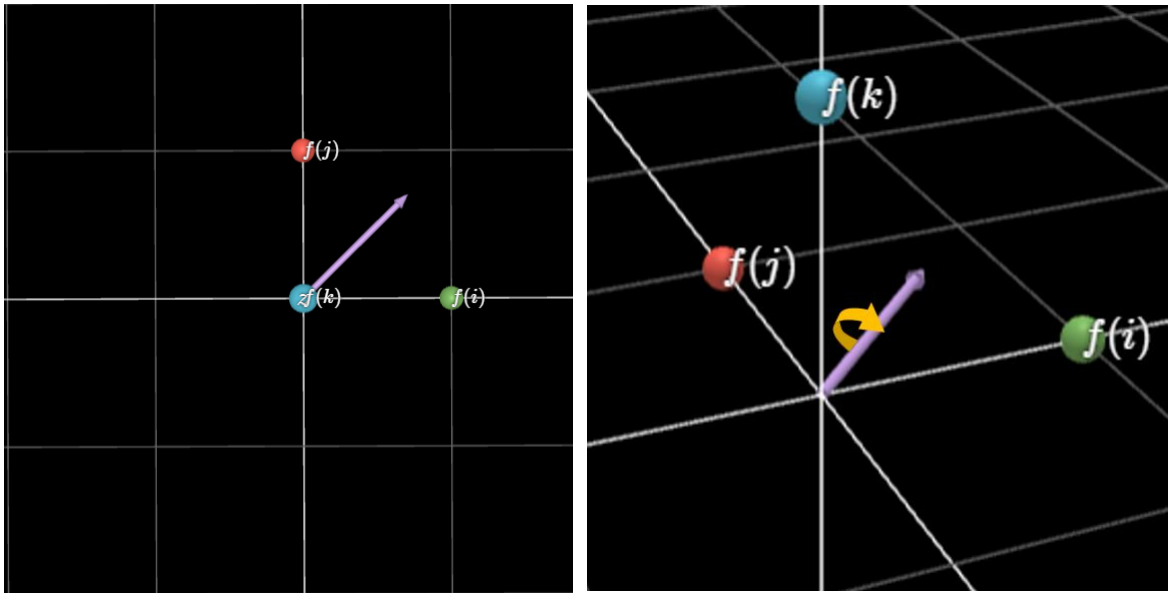

Figure 5: Rotation axis given by quaternion with b≅0.71 ,c≅0.70. Source: https://eater.net/quaternions

Dual quaternions are an extension of quaternions, represented as a pair of quaternions: one for rotation and one for translation. Mathematically, we can write a dual quaternion as

$$\bar{q} = q_r + \varepsilon q_t$$

where q_r and q_t represent the rotation and translation of an object in 3D space respectively, and ε is an infinitesimal quantity with the property $\varepsilon^2 = 0$.

Let $q_r = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)\vec{u}$ be the rotation quaternion, and $q_t = t_1 i + t_2 j + t_3 k$ the translation quaternion, constructed by the translation vector $\vec{t} = t_1 + t_2 + t_3$. A rigid transformation is given by the dual quaternion:

$$\bar{q} = q_r + \frac{\varepsilon}{2} q_t q_r$$

Using the aforementioned concepts, we can construct an efficient representation for 3D rigid body transformations, allowing for the extraction of effective features related to the structure of the skeleton and the style of its motion, in the context of deep learning.

3.2.2 From Euclidean Transformations to Dual Quaternions

The databases used to train our networks store motion capture data in the BVH file format. The Biovision Hierarchy (BVH) format stores a hierarchical skeletal structure and positional & rotational information as 3D coordinates and euler angles (within a coordinate system with the origin usually located at the Hip joint) respectively. The data in the motion segment can be easily represented by traditional transformation matrices. Since our network assumes poses represented as dual quaternions, we need a mechanism to convert eulerian angles and positional information to dual quaternions. Andreou et al. describe such a mechanism, by which we can utilize mathematical equations to achieve the conversion.

Given some rotation of α radians along the x-axis, we can construct the corresponding quaternion:

$$q_{rx} = \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right) i + 0j + 0k$$

Similarly, we can construct the quaternions qry,qrz for rotations of β,γ radians along the y and z axes respectively as:

$$q_{ry} = \cos\left(\frac{\beta}{2}\right) + 0i + \sin\left(\frac{\beta}{2}\right) j + 0k$$

$$q_{rz} = \cos\left(\frac{\gamma}{2}\right) \pm 0i + 0j + \sin\left(\frac{\gamma}{2}\right) k$$

The quaternion qr which describes the orientation is computed as (assuming a zyx rotation order):

$$q_r = q_{rz} \cdot q_{ry} \cdot q_{rx}$$

The dual part qd, which represents the displacement, is obtained by:

$$q_d = \frac{1}{2} q_t q_r$$

$$q_t = 0 + xi + yj + zk$$

where *x,y,z* denotes 3D displacement in Cartesian coordinates.

3.3 Skeleton Aware Operations

Very often, the encoder & decoder units in an autoencoder network will be convolutional neural networks instead of multilayer perceptrons. For example, in the case of an autoencoder network in the context of some computer vision problem, it makes sense to include convolutional layers, so as to extract useful features from the input while encoding it. Those features are then utilized in the decoder, where the latent space gets "upsampled" back to the dimension of the original input. If - for instance - we encoded an image into some reduced, latent representation, and then attempted to decode it by simply upsampling/increasing the resolution, it would be nearly impossible to reconstruct something that closely resembles the original input without any additional information. The "additional information" required, would be the features extracted from the learned filters in the encoding phase.

Similarly, in the context of computer animation, we require an operation that yields useful features from an animation. Aberman et al. [6] propose a novel motion processing framework that incorporates skeleton aware operations, namely: skeletal convolution, skeletal pooling, and skeletal unpooling.

Given a skeleton with J joints represented as a list $J = \{j_0, j_1, \dots, j_J\}$, the joints' hierarchical structure can be represented with a list of the same length that contains the index of each

joint's parent $P = \{p_0, p_1, \dots, p_J\}$. For each joint with index *x*, its neighbors N_x is a set of joints, that when interpreting the skeleton as a graph, are at a distance less or equal to

$$N_x = \{j_y \mid dist(j_y, j_x) < d\ , 0 \leq y \leq J\}$$

3.4.1 Skeletal Pooling & Unpooling Operation

A skeleton is pooled by collapsing pairs of consecutive joints, and unpooling is the reverse procedure. Consider the scenario where the goal is to arrive at some reduced version of the human skeleton, containing the root joint and the 5 end joints (head, hands, feet). We would need to iteratively remove joints between the root and one of those 5 end joints. The skeleton after removing n joints between the root and the end joints is the nth pooling level.

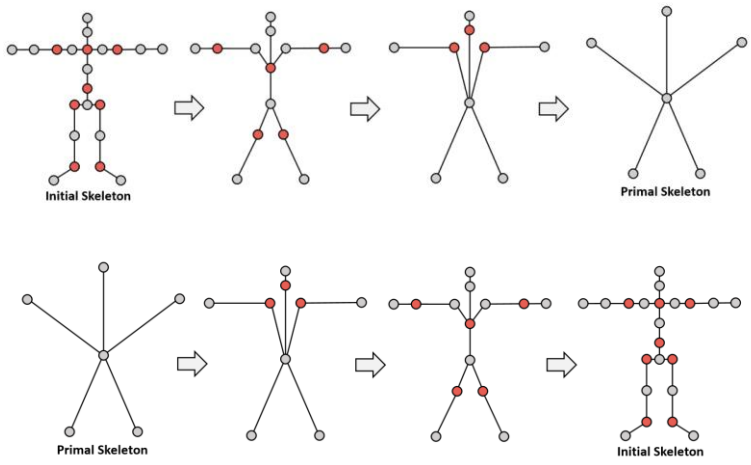

Figure 6: Skeletal Pooling (top) and unpooling (bottom) operations. Source: https://arxiv.org/pdf/2311.02191

3.4.2 Skeletal Convolutions

The skeletal convolution is applied as a standard one-dimensional convolution over the temporal channel at each pooling level i. Consider the rotations of a pose represented as a $Q \times J \times T$ matrix, where Q is a quaternion representing a rotation, J is the number of joints, and T is the temporal length (e.g. number of frames in a computer animation). Then, we can include an $S \times J$ matrix, where S is the static feature vector (x,y,z values for offset of some joint), and J is again the number of joints. We can perform a meaningful convolution on a motion, by convolving a joint with a filter, across some temporal length while considering neighboring joints. Intuitively this can be understood as convolving "sections" of the two matrices as described above with a learned convolution kernel.

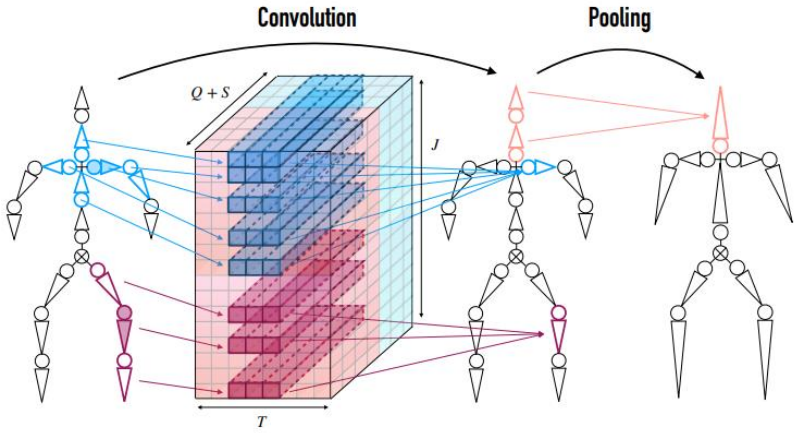

Figure 7: Convolution of a skeleton's joints with learned filters (blue & purple). The right part of the image portrays the pooling of two consecutive joints. Source: https://arxiv.org/pdf/2005.05732

3.4 Background Review

The three key concepts discussed above serve as the foundation to our Motion-Retargeting framework which is based on an Autoencoder Network with Skeleton Aware Operations applied to poses represented by Dual Quaternions. In the following section, we will illustrate our approach to designing this framework.

Chapter 4

Methodology

4.1 Overview:

In this chapter, we will describe the structure, architecture, and training process of the base framework which facilitates motion reconstruction, then, we will explain the process of acquiring corresponding motions between domains, and finally, showcase two separate methods of achieving motion retargeting. More specifically, we will discuss the incorporation of a loss function between latent spaces with the goal of achieving a system which “encourages” an autoencoder network to encode input in a particular way such that if the input example corresponds to a motion in some other species’ domain, the two encoded latent spaces will be similar enough to facilitate motion retargeting. The second method we will discuss is concerned with the introduction of an external neural network that explicitly learns to map one latent space representation to another.

4.2 Base Network Structure:

As we mentioned previously, the base for our framework is the generator network as described in SparsePoser [5].

The network takes as input a set of motion sequences (BVH format) of length T (frames) using a skeleton with J joints.

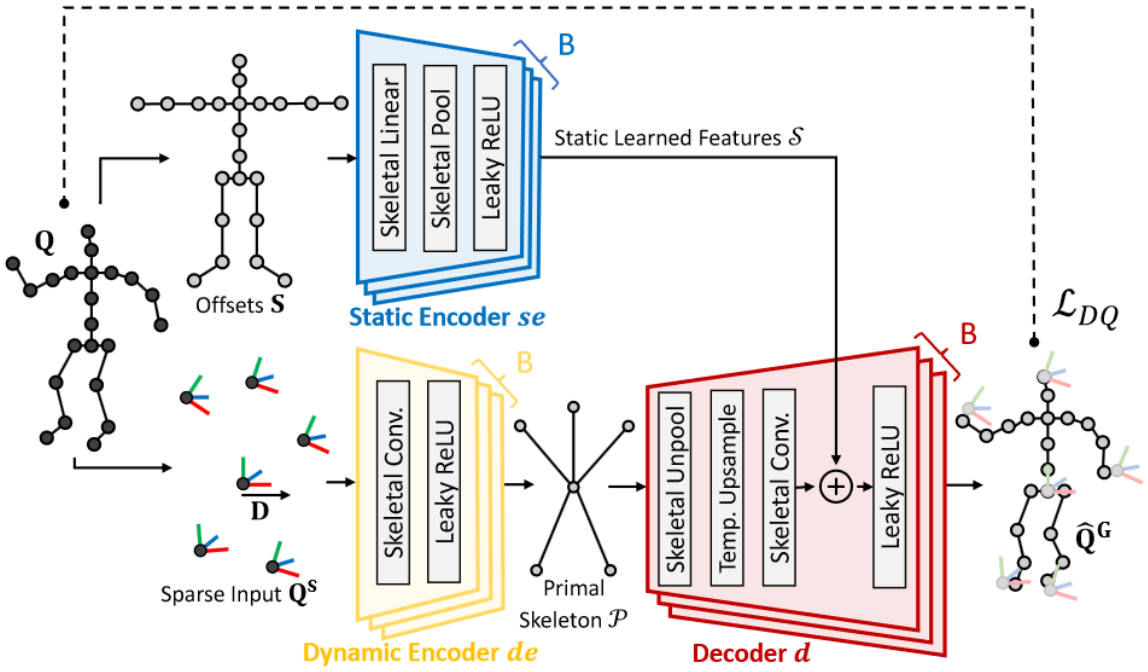

Figure 8: SparsePoser generator network (autoencoder). Source: https://arxiv.org/pdf/2311.02191

The generator has the structure of an autoencoder network, with the encoder & decoder units consisting of a series of skeleton aware layers [6]. The generator takes the components S,Q, and D as input, where $S \in \mathbb{R}^{J\times 3}$ is the static component (set of offsets that represent the local positions of the joints relative to the root joint), $Q \in \mathbb{R}^{T\times J\times 8}$ is the dynamic component (rotations and translations of all joints for every frame in the motion represented by dual quaternions), and $D \in \mathbb{R}^{T\times 3}$ is the displacement component (displacement between frames of the root joint for all frames).

The static component *S*, is fed into the Static Encoder *se*, which uses *S* to produce a list of *B* static learned features $SF = (SF^0, SF^1, \ldots, SF^B)$ where *B* is the number of pooling levels.

$$SF = se(S)$$

The Static Encoder is made up of *B* consecutive blocks where each block contains the following layers:

1. Skeletal Linear operation (Skeletal Convolution without temporal/spatial considerations)
2. Skeletal Pooling operation
3. Leaky ReLU activation

Thus for $B = 3$ a forward pass through the Static Encoder looks like this:

$$S \rightarrow se\ block_1 \rightarrow se\ block_2 \rightarrow se\ block_3 \rightarrow SF$$

The Displacement Component *D* and a subset Q^s of *Q* which contains only the sparse input (head, root, toes) is fed to the Dynamic Encoder *de* to be encoded into the primal skeleton *PS*:

$$PS = de(D, Q^s)$$

The Dynamic Encoder is made up of *B* consecutive blocks where each block contains the following layers:

1. Skeleton Convolution operation (stride of 2)
2. Leaky ReLU activation

Forward pass through Dynamic Encoder for *B = 3:*

$$(D, Q^S) \rightarrow de\ block_1 \rightarrow de\ block_2 \rightarrow de\ block_3 \rightarrow PS$$

It's noteworthy to mention that since the Skeletal Convolution is implemented as a PyTorch *conv1d* convolution with a stride of 2, the temporal dimension is halved with every pass through each block. This is because as the filter slides across the input data it skips positions, and as a result the number of output elements produced is halved.

The decoder *d* takes the primal skeleton *PS* as input and reconstructs the full body pose after adding the static features *SF* provided by the Static Encoder to the results.

$$Q^G = d(SF, PS)$$

The Decoder is made up of B consecutive blocks where each block contains the following layers:

1. Skeletal Unpooling operation
2. Temporal Upsampling (to arrive at original temporal dimension)
3. Skeletal Convolution operation
4. Leaky ReLU activation (after adding *SF* to the convolution result)

Forward pass through Decoder for *B = 3*

$$(SF, PS) \rightarrow d\ block_1 \rightarrow d\ block_2 \rightarrow d\ block_3 \rightarrow Q^G$$

The autoencoder's reconstruction loss is computed as:

$$L = MSE(Q^G, Q)$$

Where Q^G is the reconstructed motion, and Q is the dynamic component of input motion.

4.3 Data Augmentation

During the development of this thesis, it became apparent that data augmentation is necessary for the successful training of a network which can retarget motions across the different domains. The reason for this is twofold:

1. Existing animal motion databases are limited in size. This poses a problem for a data-driven network such as the one we are using for the base of our framework, which relies on extensive training data for producing high quality results.
2. It is necessary to incorporate sufficient pairs of "corresponding" motions within the training dataset. We define corresponding motions between two domains as motions that can be described/labeled as the same activity, for instance an animation of a dog walking, and one of a human walking, comprise a pair of corresponding motions.

Further elaborating on the second point, the necessity for pairs of corresponding motions stems from the design of the two methods we propose for achieving motion retargeting, which we will delve into further in the next section.

The two techniques we apply to augment our databases are animation mirroring and motion synthesis using generative models.

4.3.1 Animation Mirroring

To mirror an animation, we flip everything along the X axis, effectively creating a reflection of the original motion. Animation mirroring not only doubles a dataset, but also ensures that the left and right sides of a skeleton are "equally represented" in the data.

4.3.2 Motion Synthesis

To synthesize new data, we use GANimator [4] which works as described in section 2.2.1. Using a python script, we extract a motion sequence from the original dataset we wish to extend. A few hundred frames (~10 seconds) is all that is needed to obtain high quality results from GANimator.

We opt for simple - repetitive motions like walking and running for two reasons. First, motions like walking and running are easy to locate across datasets of different domains, thus training GANimator on such motions allows us to generate practically infinite examples for which we can find corresponding motions. If we instead opted to train the generator network on an unusual example such as a human performing a backflip, the obtained generator would be of less value to us, as the synthesized motions would likely not be pairable to a corresponding motion from another domain. The second reason we opt for simple repetitive motions, is to ensure high quality synthesized motions. By feeding the network with several examples of real poses, we are less likely to get unrealistic results or motions with artifacts. Even though we don't measure the error of the produced motions quantitatively, we still require the results to be plausible and look believable.

4.4 Motion Retargeting

Our main contribution includes two distinct methods that extend the capabilities of the SparsePoser generator so as to facilitate motion retargeting between different domains (structurally dissimilar species).

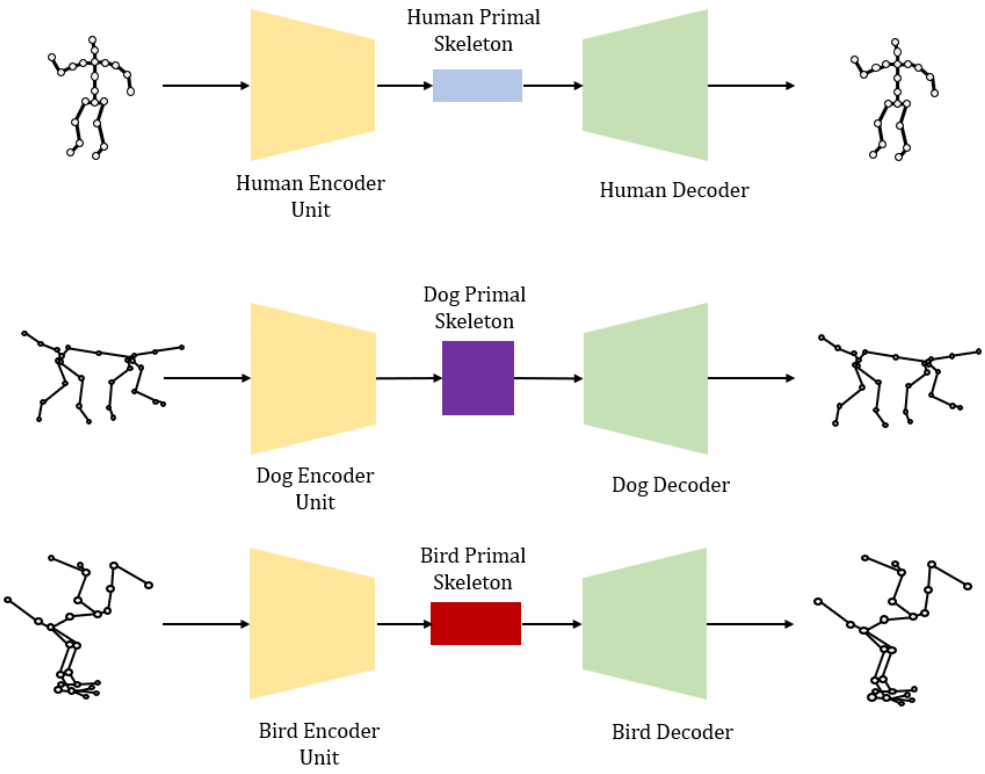

Figure 9: Human (top), dog (middle), and bird (bottom) autoencoder networks. The three encoders produce entirely different latent representations (primal skeletons).

4.4.1 Shared Latent Space

As we previously explained, the first unit in the autoencoder network we use for motion reconstruction comprises an encoder which reduces input data (motion) to a primal skeleton (latent representation), which is composed of the root joint (hips) and the end effectors. As seen in related literature [6], distinct autoencoder networks trained on data of different yet structurally similar skeletons, are able to facilitate retargeting without the help of additional

techniques, as the encoders of said networks will produce very similar latent representations for corresponding inputs (by virtue of being trained on similar skeletal structures). This means that given N autoencoders, having all networks trained on different but structurally similar skeletons, they can configure an any-to-any conglomerate of networks in which it is possible to feed an input motion into one encoder and obtain the reconstructed motion from any or all other decoders.

In order to harness the power of the shared latent space in our case where we handle substantially dissimilar skeletal structures, two conditions must be met. First, the dimensions of the latent representation of two domains for which we wish to allow motion retargeting, must match. Second, we must ensure that corresponding motions between domains will indeed produce similar latent space representations, such that the retargeting can be achieved by passing the primal skeleton obtained from an encoder trained in domain A, to a decoder trained in domain B.

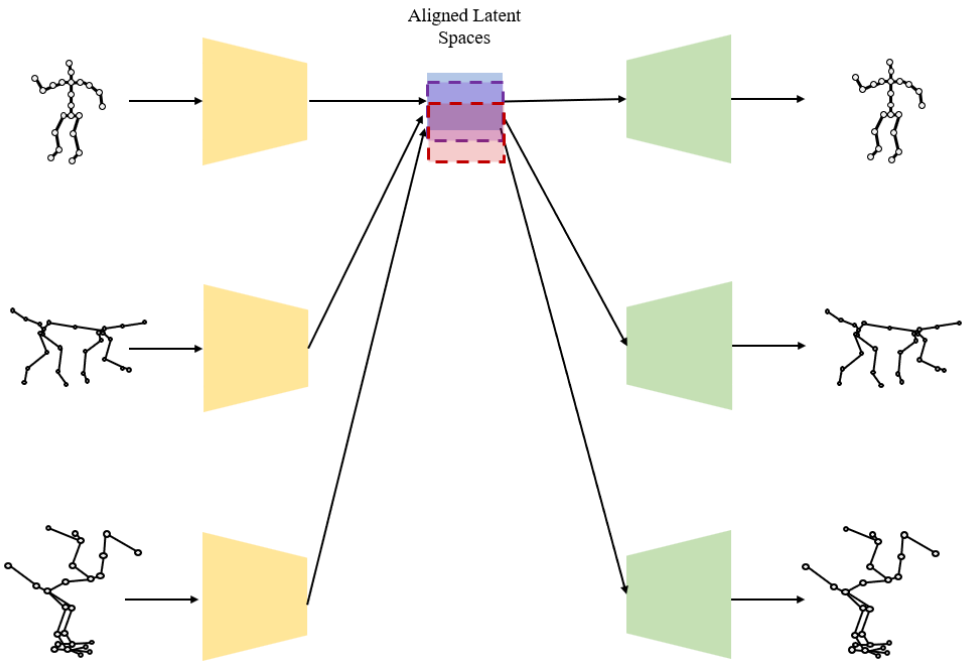

Figure 10: Human, dog & bird autoencoders after latent space alignment.

4.4.1.1 Latent Space Dimensionality

What we essentially aim to do is use the output of the encoder unit of some autoencoder A, as the input to the decoder of some separate autoencoder B. This imposes the structural constraint of having encoders/decoders which are compatible with each other across different networks. In our case, the output of the encoder is a 3D tensor with a shape of:

$$\left[batch\ size\ , num\ of\ sparse\ joints \times 2^{B}, \frac{num\ frames}{2^{B}}\right] \mid B = num\ of\ poolings$$

The important parameter here is the number of sparse joints, which is dependent on the initial structure of the skeleton. The primal skeleton will contain the root joint + all end effectors, so for a human skeleton, that would be 6 joints (root,head,hands,feet). Many animals however also have tails, which is an extra end effector, making the shape of the primal skeleton (and thus the latent space representation) different. So long as two networks have structurally incompatible encoders and decoders, motion retargeting by means of shared latent space is impossible.

4.4.1.2 Dummy Joint

One way to overcome this hurdle, is to modify the initial structure of certain skeletons to include extra "dummy" joints, so as to make the number of end effectors equal to that of the others. If we make a concession that our framework will support domains with up to 6 end effectors (7 sparse joints including root joint), then the human primal skeleton becomes structurally compatible with most quadrupeds (dogs, cats, etc.) by adding a single dummy joint to it.

We can implement this dummy joint by modifying the hierarchy of the BVH files in the human motion database. We define a new joint at the end of the hierarchy, with the same offset and rotations as the root joint. Effectively, this means that the human database will have two identical sparse joints, one being the root, and the other the dummy joint. Similarly

we could add more dummy joints for skeletons that have even less end effectors, in order to reach our assumed maximum of 7 sparse joints.

While this does solve the issue of latent space dimension matching, it introduces a new problem, this time in the training phase. Specifically, modifying the skeletons to include extra end effectors increases the size of the network, and consequently the amount of parameters. Adding a single dummy joint to the human database increases the autoencoder's already large parameter count by 15%. Larger neural networks can have difficulties converging for several reasons. Very often we observe larger neural networks being more susceptible to overfitting as the models become too complex relative to the available training data. Also, due to the fact that we feed the autoencoder the same information twice (the dummy joint is a copy of the root joint), the network may have a tougher time converging as a result of less informative gradients.

4.4.1.3 Removing Joints

A different way to address the problem of latent space dimensionality matching, is by removing joints from skeletons which have more end effectors than humans. For instance: removing the tail part from skeletons which include tails. Using this method, our concession would be not for an assumed maximum number of sparse joints, but a minimum instead. Considering that our goal is to provide a framework that effectively utilizes the human skeleton as a motion driver for generating motions in other domains, it makes sense to place that minimum number of sparse joints at 6 (hip joint, head, hands, feet). Thus, in order to achieve cross domain network compatibility, we would need to remove end effectors from skeletons until the remaining amount of sparse joints is equal to our minimum of 6.

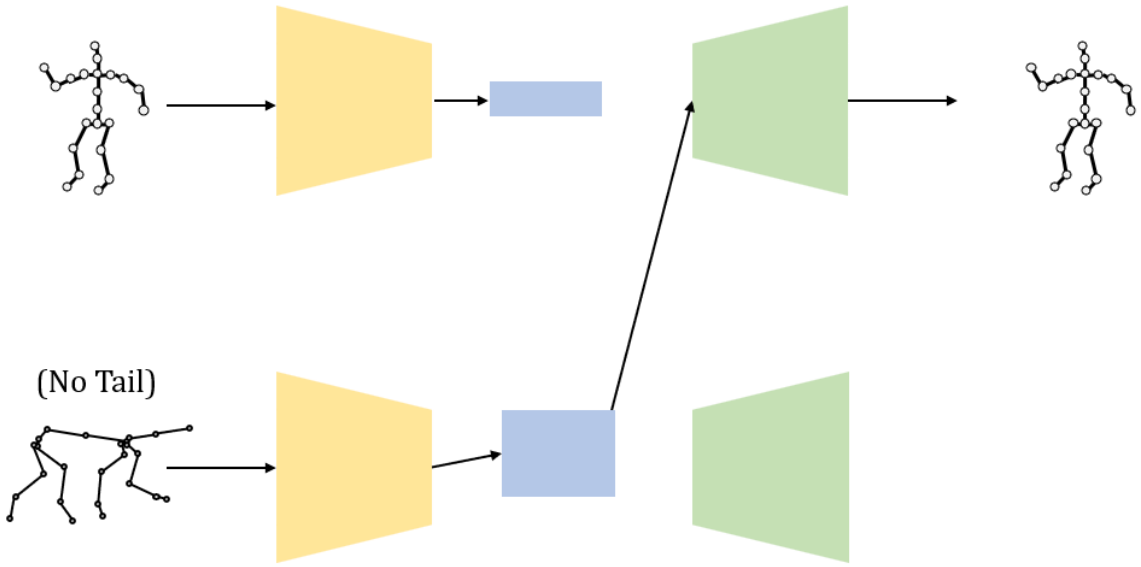

Figure 11: Dog motion sequence with the tail joints removed so as to match the dimensions of the human primal skeleton. Notice how the dog latent representation is able to be passed through the human autoencoder, however since they are still semantically different, the result is expected to just be noise.

Immediately, an obvious problem with this approach is that information is being lost. We are trading off information for cross-domain network compatibility. A justification for this trade off would be that the positional information of a body part such as the tail of a dog, might be insignificant to the rest of the motion, especially in cases where the movement of the tail is dictated by external stimuli (positive/negative feelings). In other cases though, the tail's motion might be directly related and inferable from the rest of the body's motion [10] (e.g. counteracting body motion for balance).

This observation might also pose a solution for the above problem of information loss. We propose a slight modification to the SparsePoser framework that looks to utilize the fact that sometimes a joint's movement is inferable from other joints movement.

As explained in section 4.2, the encoder unit in SparsePoser is made up of a Dynamic Encoder *de* and a Static Encoder *se*. Considering a motion M, and its version with some limb

removed M’ (e.g. M’ is M with tail removed), we can extract the static components S and S’, and the dynamic components Q and Q'. Feeding Q'^{s} (sparse inputs) into *de*, and S into *se*, we obtain a latent space representation that has dimensions compatible with those of the assumed minimum, while keeping the static learned features from the original skeleton. Then, by slightly modifying the decoder network *d* to consider the original structure of the skeleton while performing the unpooling operations and adding the static learned features SF from *se*, the decoded result Q'^{G} has the same skeletal structure as M (i.e the tail is included). To make this version of SparsePoser actually able to function, a final modification is needed, this time on the reconstruction loss which is being computed as $L = MSE(Q'^{G}, Q')$. Instead, the new reconstruction loss must be:

$$L = MSE(Q'^{G}, Q)$$

i.e, we try to minimize the loss between the decoded result Q'^{G} and the motion from the original motion data M (Q), instead of the input M’ (Q'). The core idea behind this proposition stems from the fact that the missing joints’ movement can be derivable from the movement of the rest of the joints. Our only requirement is for the reconstructed motion to look plausible, hence, the case in which for some reason the missing joints’ movement is in fact not inferable from the rest of the motion, the reconstructed result needs only to look believable.

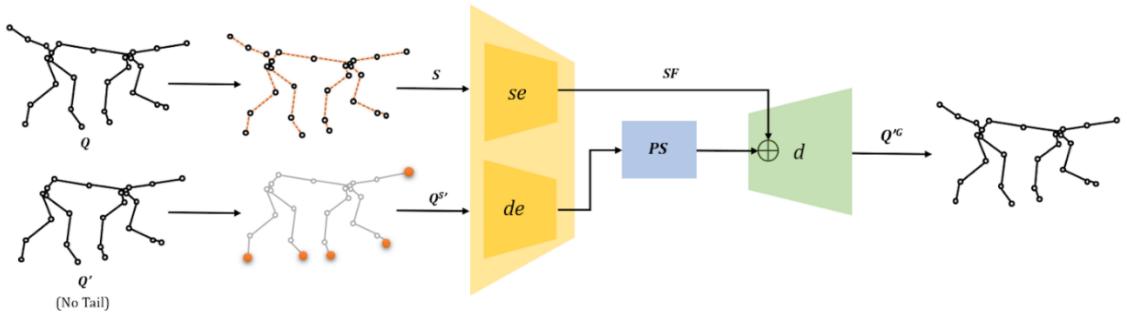

Figure 12: Dog motion sequence with the tail removed is fed into the dynamic encoder, while the original skeletal structure (static features) is fed into the static encoder.

4.4.2 Latent Space Loss Minimization

After successfully manipulating our data with either of the two aforementioned methods and obtaining the modified databases that allow for cross-domain network compatibility as explained in section 4.4.1.1, we can encode motion in some domain, and decode it in any or all others. As previously stated, motion retargeting refers to the transfer of motion from one domain to another while preserving the characteristics. The goal here is to encode a motion from a domain *D* into a primal skeleton, and then decode it in another domain *D'* and have the decoded motion be one that corresponds semantically to that of *D*, but with the appropriate style and nuances of *D'*. However, running the encoded motion (primal skeleton) from domain *D* through a decoder from domain *D'* does not guarantee an accurate reconstruction at all. In fact, we would expect to see a very noisy result. This is due to the fact that the latent space representations for dissimilar skeletal structures is expected to be different enough to not accommodate motion retargeting without additional adjustments.

Our proposed solution to this problem is to attempt to obtain encoding functions which produce similar latent spaces for corresponding motions, in the training phase of the autoencoder. In order to achieve this, we introduce a Latent Space Loss defined as

$$L_{latent} = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$$

Where x_i and y_i are the ith elements of the vectors representing two latent space encodings from 2^B frames of motion, and B is the number of pooling levels as described in section 4.2. This is known as the Euclidean Distance Loss. We chose this loss function as it closely matches the intuitive understanding of latent space dissimilarities (geometric dissimilarities).

The latent space loss would work in congruity with the reconstruction loss, together making up the total loss we are trying to minimize.

$$L_{total} = L_{latent} + L_{reconstruction}$$

Two questions arise from the above loss function suggestion. The first question is in regard to how exactly we should measure latent space dissimilarities between two domains, i.e. for which motion examples do we choose to minimize the loss over. The second question, is how do we avoid the scenario where the model is too focused on the latent space loss, to such an extent that the reconstruction loss is never minimized adequately (or vice-versa)?

On the second question, our current solution is to assign weights to the two losses, and fine-tune the values until an adequate model is trained.

$$L_{total} = w_1 L_{latent} + w_2 L_{reconstruction}$$

Concerning the first question, we suggest a manual mechanism by which the autoencoder is explicitly "told" which training examples to consider for the latent space loss. This consists of labeling training examples for which corresponding motions in other domains exist. As we mentioned previously, the human skeleton acts as the motion driver for reconstructing motions. As such, when training an autoencoder instance on a new domain, we will attempt to compute the latent space loss over corresponding motions in the human domain only. By selecting this approach where we optimize the loss function one-sidedly (only modifying the network of the added domain while keeping the human network static), we simplify the

process of aligning latent spaces by encouraging all latent representations to “look like” the human primal skeleton. Contrast this with an approach in which you try to minimize the loss by attempting to modify all networks in order to converge to some “global” primal skeleton, which would mean introducing a new domain to the network conglomerate means having to re-train all autoencoders.

human_motion_1.bvh	dog_motion_1.bvh
human_motion_2.bvh	dog_motion_2.bvh
human_motion_3.bvh	dog_motion_3.bvh
…	…
human_motion_corresponding_1.bvh	dog_motion_corresponding_1.bvh
human_motion_corresponding_2.bvh	dog_motion_corresponding_2.bvh

Table 1: Example of a human & dog mocap dataset, with the last two files being corresponding motion pairs.

The steps to this mechanism are as follows:

1. Train autoencoder for human domain normally
2. For a new domain, locate motions which can be paired with examples from the human domain as corresponding motions and label them appropriately.
3. If needed, train a generative model such as GANimator on such motions and generate even more examples of corresponding motions
4. The new domain’s autoencoder will compute and try to minimize the latent space loss function only for the corresponding motions, while for the rest of the training data it remains frozen.

On point 4, we must note that it's necessary for a pair of corresponding motions to be of the same length (number of frames), therefore some further processing of the data will probably be necessary (cropping motions to obtain corresponding pairs of same lengths). Additionally, this is the point where we run into the problem of phase and velocity matching.

By velocity, we mean the speed at which a skeleton's root joint moves through space. It is not a given that two corresponding motion examples (e.g. dog walking & human walking) of the same temporal length (frames) will have matching velocities. By phase, we mean the point at which a repetitive motion (e.g. walking) is within its cycle. Different species have differing stride lengths, thus two skeletons of different species or sizes will be out of phase even when traveling the same distance. For instance, a 1 meter forward movement from a dog might take 4 strides, but only 2 for a human. We propose mutually exclusive solutions to these problems. If we wish for a retargeted motion to maintain the velocity of the original input, then we cannot ensure phase matching, and vice versa.

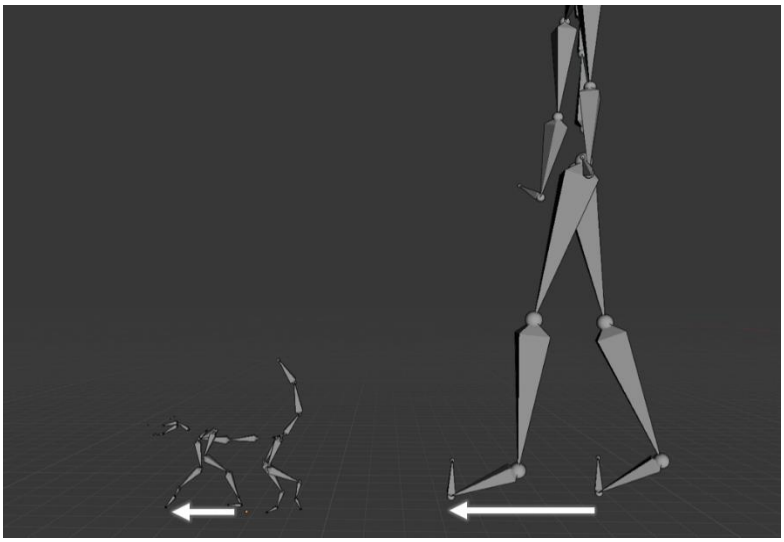

Figure 13: Visual explanation of difference in stride length between cat (left) and human (right).

Figure 13 makes the incompatibility of phase and velocity in different species easier to comprehend. It's evident that due to the cat's smaller stride length, it covers less distance than the human for the same number of strides/steps. Thus, if we want a retargeted motion to maintain the input motion's velocity, the movement's phase will likely not match. If we

instead want to preserve the phase from the original motion, we are sacrificing matching velocity.

4.4.2.1 Velocity Matching

To achieve motion retargeting while maintaining the source velocity, we calculate the root joint's velocity vector for two consecutive frames, at frame i as

$$\overrightarrow{u_i} = \vec{r_i} - \overrightarrow{r_{i-1}}$$

Where ri,ri-1 are the root joints position vectors at frames fi and fi-1 respectively. We calculate the root joint's velocity vector for a window of frames as

$$\vec{u} = \left(\overrightarrow{r2} - \overrightarrow{r1}\right)/\Delta f$$

Where Δf is the number of frames in the window.

We then pass $\vec{u}$ to the network as a learned parameter and incorporate a loss function for it

$$L_i^{per\ frame\ velocity} = MSE\left(u_i^{retargeted}, u_i^{source}\right)$$

$$L^{velocity\ avg} = \frac{1}{F}\sum_{i=1}^{F}(L_i^{per\ frame\ velocity})$$

$$L = w_1 L^{reconstruction} + w_2 L^{latent} + L^{velocity\ avg}$$

4.4.2.2 Phase Matching

For phase matching, we introduce two foot-contact labels (one for each human foot) and label each frame as having foot contact (1) or not (0). The labels on the input data are valued by comparing the height of the feet end effectors to some threshold *t* - If a foot end effector has height less than *t* then we say that there is foot contact. Since not all skeletons are bipedal, we have to specify which joints in the reconstruction domain will correspond to the human feet end effector joints (e.g. the hind legs of a dog).

In training, the reconstructed motion from some domain D is compared to its corresponding motion pair from the human domain for phase matching, and the model is penalized for incorrect foot contact predictions using a slightly modified binary cross entropy loss function:

$$L_i^{left\ foot} = -(y_i \cdot \ln(1 - h_i) + (1 - y_i) \cdot \ln(h_i))$$

Where h_i is the normalized calculated height for the left foot in frame i, and yi is the ground truth label for the left foot in frame i. Similarly, we construct the right foot loss function $L^{right\ foot}$. Finally, we compute the total foot contact loss as the sum of the individual per-frame losses for both feet.

$$L^{foot\ contact} = \frac{1}{F}\sum_{i=1}^{F}\left(L_i^{left\ foot} + L_i^{right\ foot}\right)$$

$$L = w_1 L^{reconstruction} + w_2 L^{latent} + L^{foot\ contact}$$

A more sophisticated phase matching technique like the one found in [12] might be used in the future instead.

So far, we have laid out our first proposed method (Shared Latent Space) for extending the SparsePoser's framework capabilities in order to facilitate cross-domain motion retargeting. This method seeks to realize motion retargeting by:

1. Satisfying cross-domain network compatibility constraint by the use of dummy joints or joint removals (latent space dimensionality matching).
2. Introducing a latent space loss.
3. Explicit informing of the autoencoder for corresponding motion pairs between domains.
4. Velocity and Phase Matching.

4.4.3 Latent Space Transfer

In this subsection, we will describe our second proposed method for achieving cross-domain motion retargeting - Latent Space Transfer. This method attempts to directly map one latent space representation onto another, by the use of an external neural network trained specifically for this purpose. The network in question would be a feed-forward neural network (FNN) made up of a series of fully connected layers. The conscious choice to not include convolutional or pooling layers was made due to the fact that we are already dealing with reduced data with extracted features (latent representation of motion), thus reducing the data even further might result in loss of valuable information. Additionally, with this approach we are relying on the geometric properties intrinsic to the latent representations (primal skeletons) to achieve a direct mapping functions between them, hence the simple network architecture.

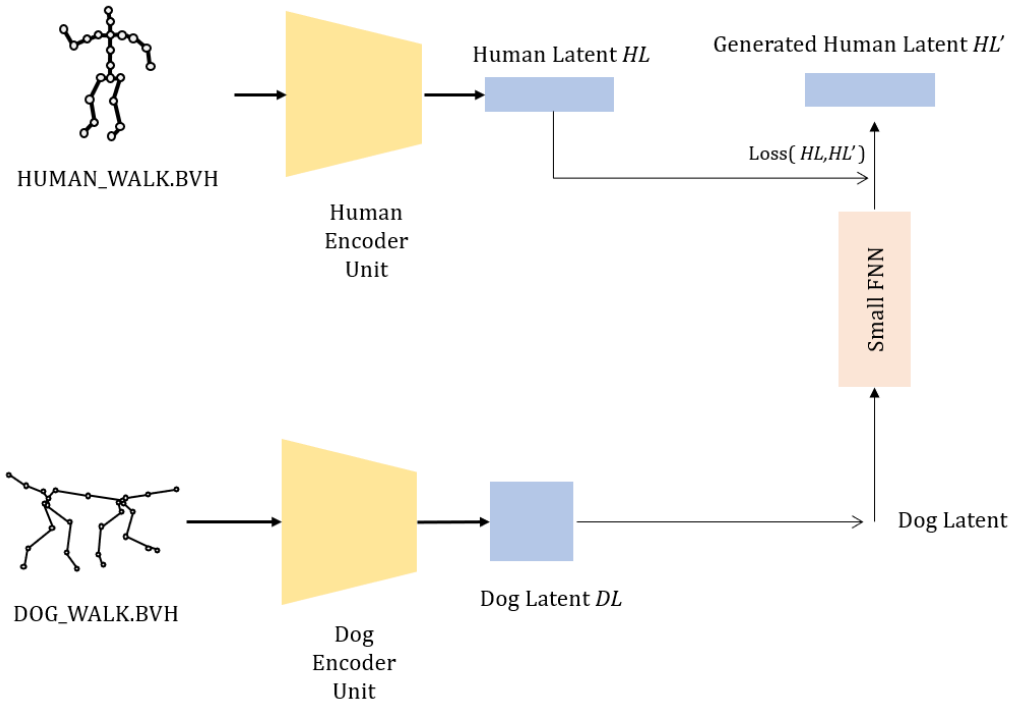

Figure 14: Training procedure for FNN. Two corresponding motions are ran through their respective encoders and the latent representation is extracted. Then the FNN learns to map on onto the other.

To train the FNN we extract the latent representations of two corresponding motions from a domain D and the human domain. We feed D's latent representation into the network which minimizes a loss function over the latent space from domain D, and the latent space for the human domain. The loss function is computed not for every frame - but for every 2^B (B is the number of pooling levels) frames, as the latent representations are temporally downsampled. This ensures that the network is not simply learning to map poses to poses, but motions to motions. The loss function used here is identical to the one we use for the latent space loss in our first proposed method for motion retargeting (Shared Latent Space).

$$L_{latent} = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$$

After obtaining the predicted mapping for the human domain, we simply feed it into its decoder and acquire the retargeted motion.

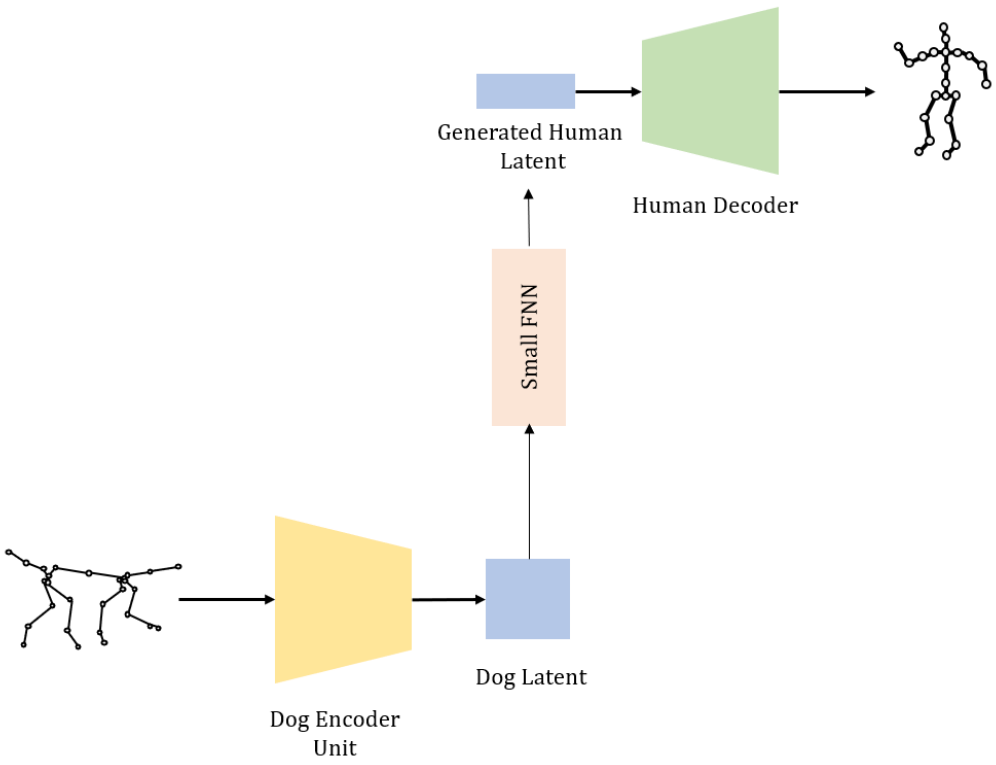

Figure 15: Achieving motion retargeting from the dog domain to the human domain by incorporating an FNN which has learned to map the dog primal skeleton to that of the human's.

Likewise, we set up a second FNN which takes the human latent space as input and outputs domain D's latent space, and proceed to train it on the same dataset. Together, these two networks provide two mapping functions (H2D and D2H) which allow us to travel from the human domain to domain D and back. If we wish to introduce a new domain D' to our system, then we simply need to train 2 new FNNs - one for the human to D' mapping, and one for D' to human. If we wish to retarget a motion from D to D', then we simply use the human domain as a sort of mediator/bridge to get there.

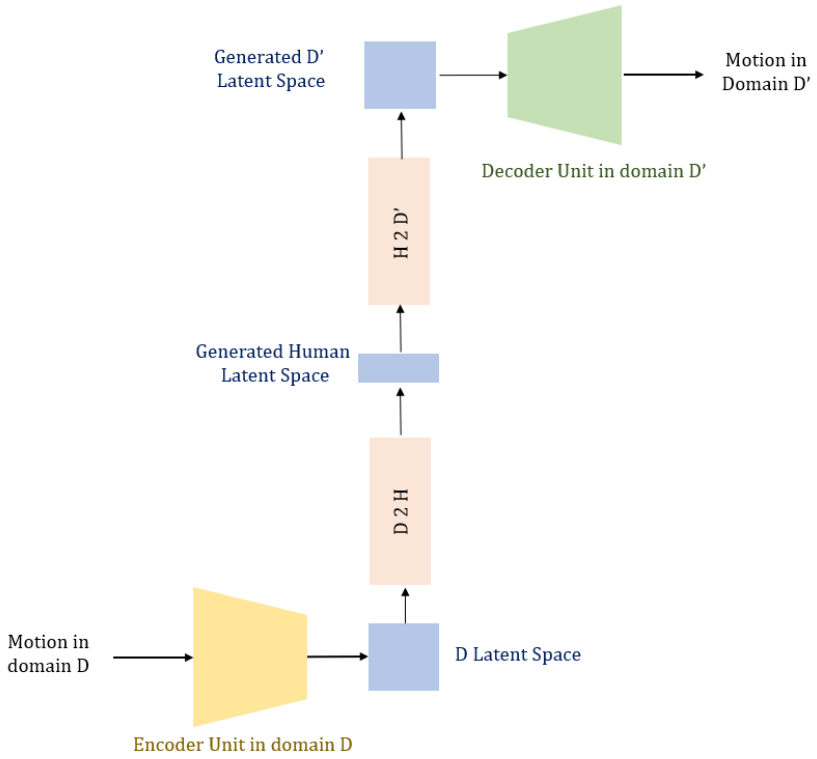

Figure 16: Network design proposal for facilitating cross-domain motion retargeting by leveraging the human latent space as an intermediary stage of getting to the target domain.

An instance of the FNN Latent Space Transfer network is comprised of the input layer, two fully connected hidden layers, and the output layer. More specifically, the network architecture is as such:

```
hidden_size = (input_size+output_size)/2
self.fc1 = nn.Linear(input_size, hidden_size)
self.relu1 = nn.ReLU()
self.fc2=nn.Linear(hidden_size, hidden_size)
self.relu2 = nn.ReLU()
self.fc3=nn.Linear(hidden_size, hidden_size)
self.relu3 = nn.ReLU()
self.fc4=nn.Linear(hidden_size, output_size)
```

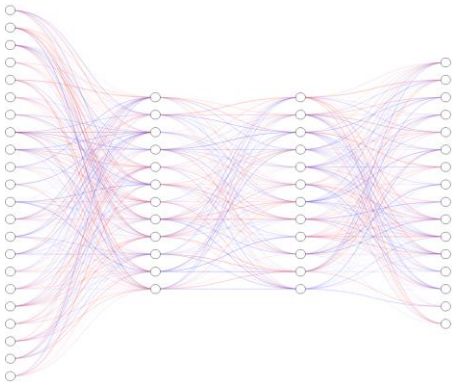

Figure 17: FNN architecture implemented in PyTorch (left). Visual representation of FNN (right).

4.5 Methodology Review

Both of our proposed approaches provide scalable methods for facilitating cross-domain motion retargeting in the SparsePoser network. A significant advantage of the Latent Space Transfer (LST) method over the Shared Latent Space (SLS) method, is that the latter requires modifications on the datasets in order to satisfy the cross-domain network dimensionality constraint (described in section 4.4.1.1), whereas the former can take a latent representation and map it to another regardless of the dissimilarities in dimension sizes. A disadvantage of LST over SLS however, is that due to its independent and external design, it cannot account for things like phase or velocity matching, whereas SLS employs the latent space loss within the autoencoder network itself during training, and has the ability to incorporate loss functions for foot contacts and root joint velocity. Additionally, LST is limited to training only on corresponding motion pairs, in contrast to SLS which is employed during the normal training stage of the SparsePoser autoencoder network. Further elaborating on this last note, even though the latent space loss function is frozen while training on non-corresponding motions in SLS, it's likely that the reconstruction loss and latent space loss will "nudge" each other in the right direction, such that a sort of feedback loop is created where optimizations to one loss function will eventually start to optimize the other one as well.

In the case of attaining an effective Cross-Domain Motion Retargeting framework using either of the two aforementioned methods, we can utilize SparsePoser's built in VR Controller support, to use the human body as a motion driver for generating novel high quality motion data.

Chapter 5

Results

5.1 Training Data

Our motion database consists of an amalgamation of different datasets containing motion capture data from different animal species including humans, felines, bovidae, canines, herbi, reptiles, and rodents.

The human dataset consists of approximately 9 hours of mocap data recorded by 9 different actors performing different activities such as locomotion, stretching exercises, playing VR Games, etc. The system used to capture movements is the Xsens Awinda [5]. In addition, the human dataset includes the entire DanceDB database [11].

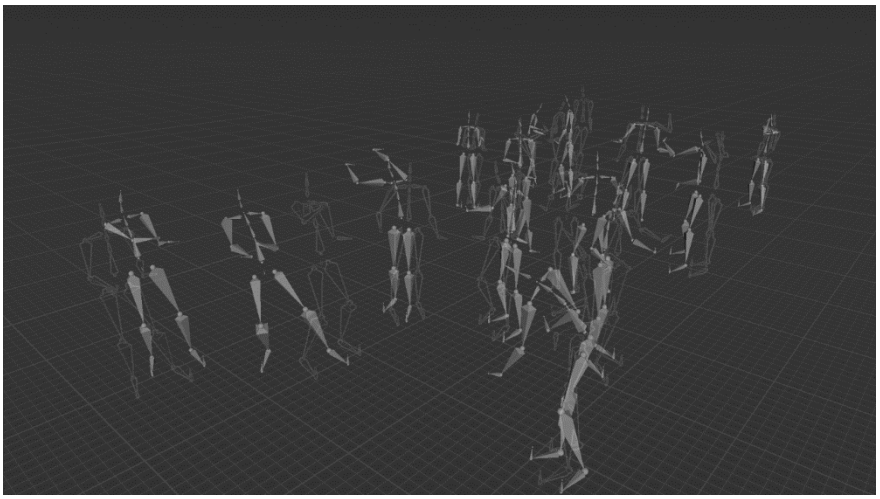

Figure 18: A small fraction of the recorded motions in the Xsens database.

The animal dataset consists of over 1 hour of motion capture footage from 73 different animals, gathered by Ellie Eliade as part of her undergraduate thesis, and classified according into the following categories: Terrestrial Limbless, Terrestrial 2-Legged, Terrestrial 4-Legged, Terrestrial 6-Legged, Terrestrial 8-Legged, Aerial 2-Legged, Aerial 6-Legged.

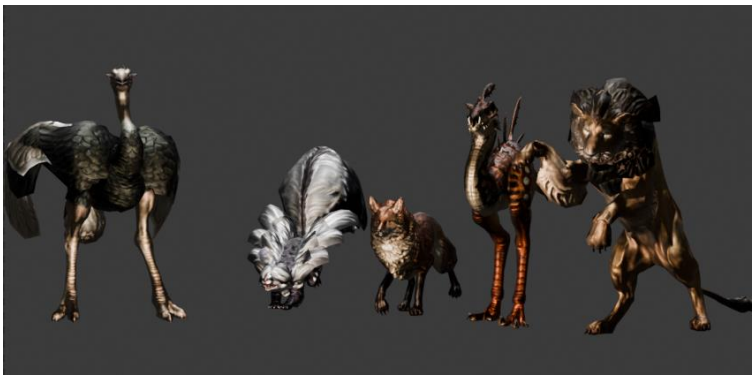

Figure 19: From left to right: Ostrich, Skunk, Fox, Raptor, Lion. Some of the animals included in the animal mocap database.

An additional 40 minutes worth of dog mocap data was added to the animal database, taken from Starke et al. [12].

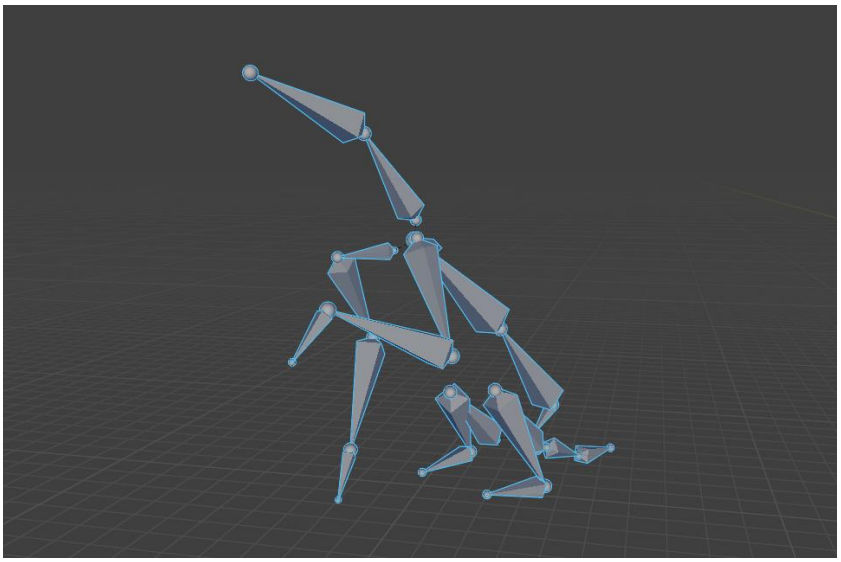

Figure 20: Dog BVH file from the Starke et al. dog mocap database.

5.2 Network Training

5.2.1 Motion Reconstruction

The SparsePoser generator network was implemented in PyTorch [13] using the AdamW optimizer [14]. It was trained with the following parameters for a human skeleton (Xsens database [5]) and a dog skeleton (Starke et al. database [12]):

```
human_param = {
    "batch_size": 256,
    "epochs": 500,
    "kernel_size_temporal_dim": 15,
    "neighbor_distance": 2,
    "stride_encoder_conv": 2,
    "learning_rate": 1e-4,
    "sparse_joints": [
        0,  # root
        4,  # left foot
        8,  # right foot
        13,  # head
        17,  # left hand
        21,  # right hand

    ],
    "window_size": 64,
    "window_step": 16,
    "seed": 2222,
}
```

```
dog_param = {
    "batch_size": 16,
    "epochs": 2000,
    "kernel_size_temporal_dim": 15,
    "neighbor_distance": 2,
    "stride_encoder_conv": 2,
    "learning_rate": 1e-4,
    "sparse_joints": [
        0,  # root
        4,  # head
        8,  # left hand
        12,  # right hand
        15,  # left foot
        18,  # right foot
        20,  # tail
    ],
    "window_size": 64,
    "window_step": 16,
    "seed": 2222,
}
```

Table 2: human parameters (left) and dog parameters (right) used in the training process of the SparsePoser autoencoder.

Due to the substantially smaller size of the dog dataset, we increase the epoch count and decrease the batch size in an attempt to allow the model to converge while simultaneously avoiding overfitting. No data augmentation was performed on either of the two datasets for the purpose of training their base autoencoders.

After thorough testing of the trained autoencoders on never before seen data, it is evident that both the dog and the human models are able to accurately reconstruct motions from never before seen inputs, such that the reconstructed motion is essentially indistinguishable from the original input.

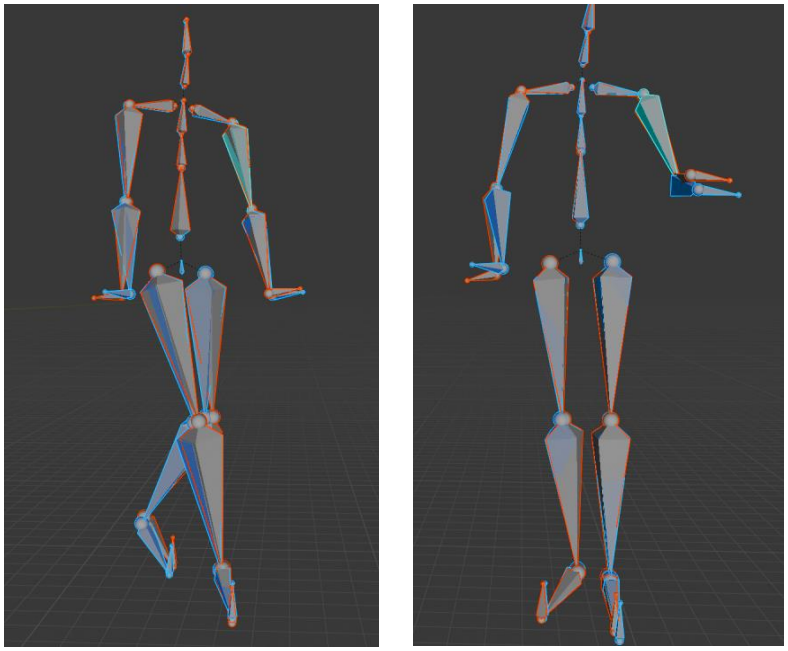

Figure 21: Reconstructed human motion (blue) overlayed on top of original input (red).

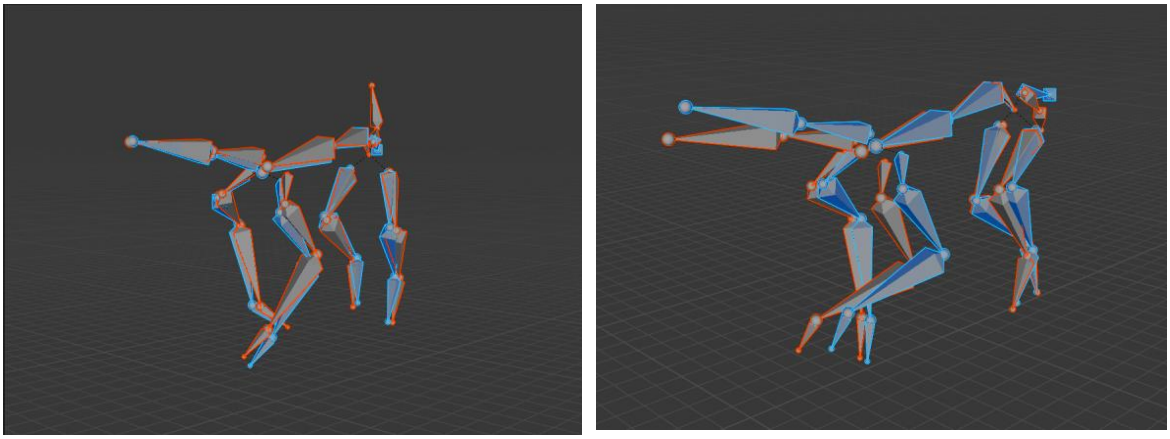

Figure 22: Reconstructed dog motion (blue) overlayed on top of original input (red).

5.2.2 Data Augmentation

We leverage GANimator's [4] capabilities to synthesize high quality novel motions, by training on single short motion sequences. We use two examples of walking sequences (one from the human dataset, the other from the dog dataset), with both of them being around ~500 frames in length. Two GANimator models (one for each domain) are then trained for

60,000 epochs each. The resulting models are able to successfully generate high quality plausible motions for arbitrary lengths, mostly without noticeable artifacts or foot-sliding. The human synthesizer is somewhat prone to foot artifacts, however more than 95% of the generated motions are of acceptable quality.

Figure 23: Input motion sequence to GANimator (top) and synthesized motion (bottom).

SinMDM [15] - a diffusion model designed to synthesize realistic motions from a single motion sequence using a lightweight shallow UNet [16] - was also used in the context of this thesis for the purpose of data augmentation, however we were not as successful with the results. SinMDM was trained on the same motion examples as GANimator, yet we failed to produce high quality with results the former. More precisely, significant positioning errors for the root joint, and phase timing issues were observed in all synthesized results.

5.2.3 Motion Retargeting

To achieve motion retargeting, we employ the Latent Space Transfer method, as described in section 4.4.3. We first extract a corresponding motion sequence pair (walking motion) from the two datasets (human & dog), and use them to train two GANimator models as described in the previous section. We then synthesize 8 different motions from the two models (4 pairs of corresponding motions) of 2000 frames each. 3 of the 4 motions in each pair were rotated by 90,180, and 270 degrees respectively around the Y axis using a script built with the PyMotion [17] library in python.

Human_walk.bvh | 500 frames	Dog_walk.bvh | 500 frames
Human_walk_generated_1.bvh | 2000 frames	Dog_walk_generated_1.bvh | 2000 frames
Human_walk_generated_2.bvh | 2000 frames	Dog_walk_generated_2.bvh | 2000 frames
Human_walk_generated_3.bvh | 2000 frames	Dog_walk_generated_3.bvh | 2000 frames
Human_walk_generated_4.bvh | 2000 frames	Dog_walk_generated_4.bvh | 2000 frames

Table 2: Set of human and dog corresponding motion pair BVH files.

Subsequently, we feed the corresponding motion pairs into their respective encoders from our pretrained SparsePoser autoencoders and extract the primal skeletons (latent space representations) for each one of the inputs.

Human_walk Primal Skeleton	Dog_walk.bvh Primal Skeleton
Human_walk_generated_1 Primal Skeleton	Dog_walk_generated_1.bvh Primal Skeleton
Human_walk_generated_2 Primal Skeleton	Dog_walk_generated_2.bvh Primal Skeleton
Human_walk_generated_3 Primal Skeleton	Dog_walk_generated_3.bvh Primal Skeleton
Human_walk_generated_4 Primal Skeleton	Dog_walk_generated_4.bvh Primal Skeleton

Table 3: Set of human and dog corresponding motion pair latent space representations (primal skeletons)

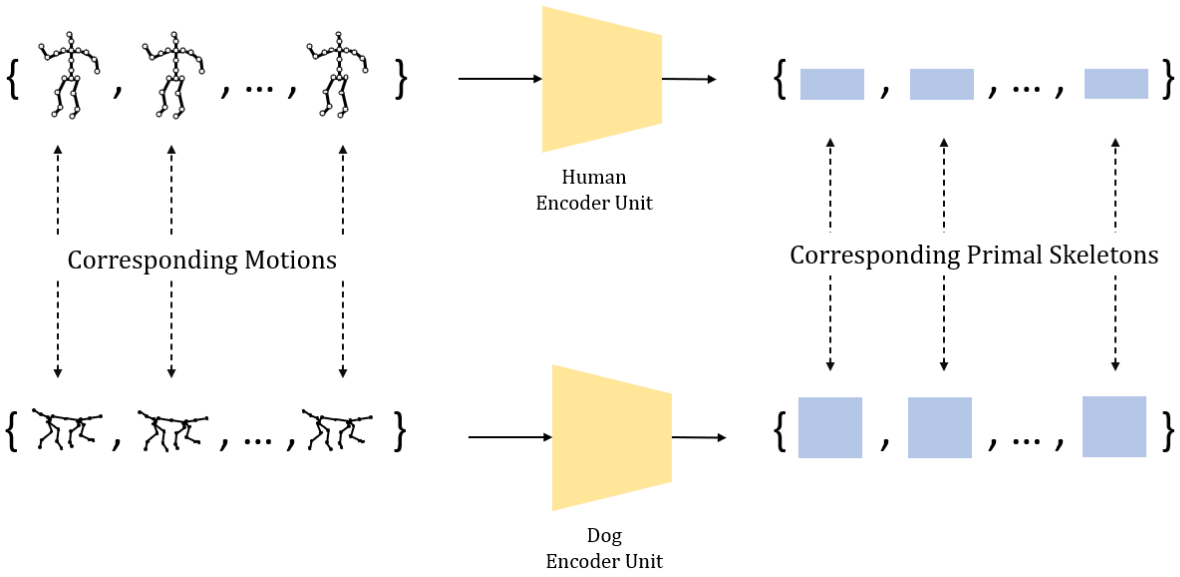

Figure 24: Process of extracting corresponding motion pair primal skeletons

We train an FNN on the primal skeleton pairs and obtain a network which has learned to transfer latent representations in the dog-to-human direction. After training the FNN for 2000 epochs, we obtain a model with acceptable performance on unseen data.

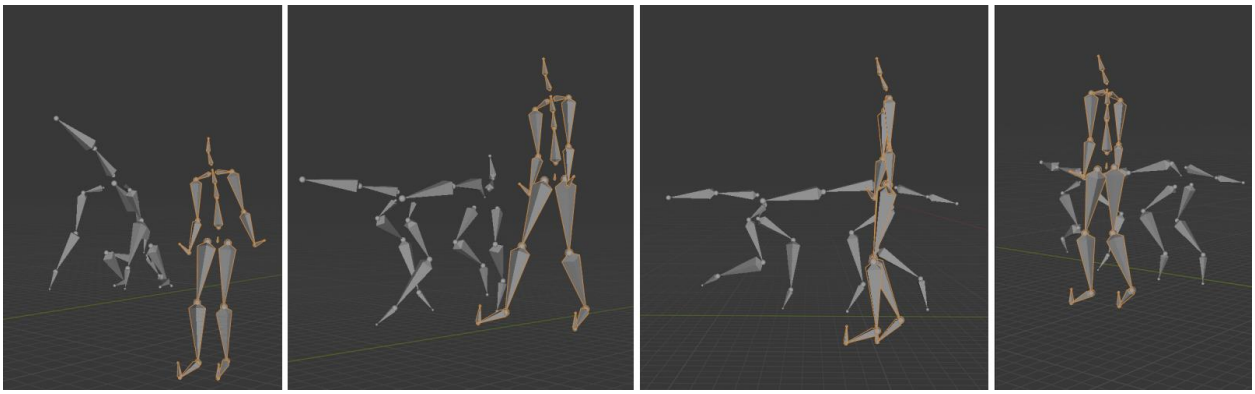

Figure 25: Latent Space Transfer method results on unseen data. The network has learned to map dog locomotion to human locomotion.

In Figure 25, we observe the performance of our Cross-Domain Motion Retargeting framework on unseen data. The FNN has successfully learned to map dog walking sequences

to human walking sequences by the method of Latent Space Transfer (as described in section 4.4.3). Even though the results look promising, especially considering the limited variety and size of the training data, we still see the FNN struggling to produce a human primal skeleton which matches the dog's sitting pose. Thus, when the predicted primal skeleton is fed into the human decoder, a sort of standing-idle position is produced instead. Additionally, the walking motion in the retargeted data is noticeably un-natural, as the left leg seems to be taking larger strides than the right – almost as if the character is limping.

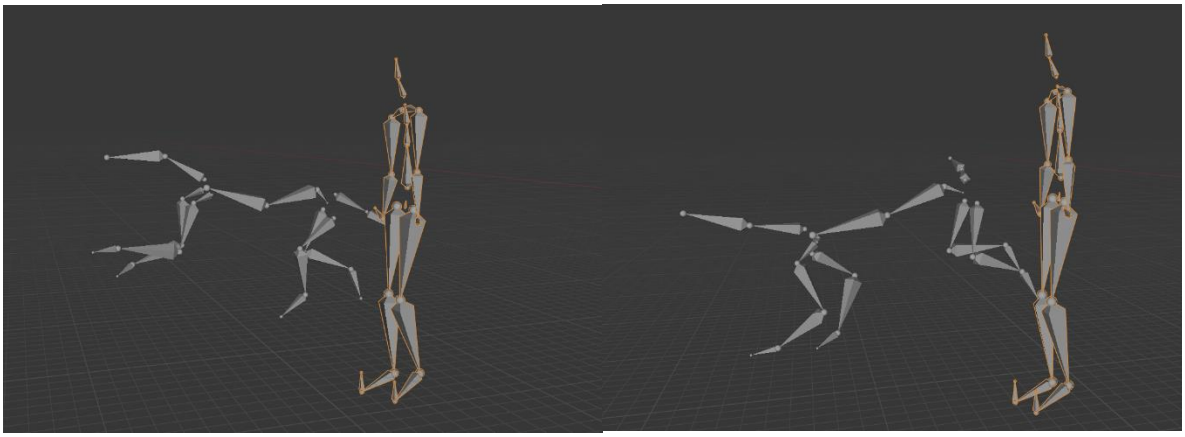

Figure 26: Results of Latent Space Transfer network on data of a dog sprinting. The resulting retargeted motion on the human skeleton performs little to no movement.

As we mentioned beforehand, the FNN was trained on paired examples of walking animations. Testing our Cross-Domain Motion Retargeting framework on inputs that are drastically different than the training data, the limitations of our FNN become apparent. We feed a sprinting animation in the dog's encoder and pass the encoded primal skeleton to our FNN (dog-to-human). The resulting predicted human primal skeleton is then fed to the human decoder, and the motion is "retargeted" (figure 26). The retargeted motion is mostly still/idle – this is a clear indication that our FNN does not generalize well, which was to be expected considering the extent of the training data.

All neural networks were trained on a system equipped with a GeForce RTX 4060-Ti 16GB VRAM GPU, Intel Core i5-13500 CPU, and 32GB RAM.

5.3 Review of Results

In this chapter, we went over the collection of our diverse motion capture database, and showcased how we effectively utilize the human and dog datasets to train two autoencoder models based on the SparsePoser framework, which can accurately reconstruct motions. Then, we explained how we make use of the GANimator framework to attain two generative models (one for each domain) that can synthesize high-quality motions, suitable for training our Latent Space Transfer FNN. Finally, we highlight the results of our Cross-Domain Motion Retargeting framework, showcasing that it performs decently on unseen data which are somewhat similar in nature to the training examples, with the prospect of generalizing even better - provided we utilize more diverse training data.

Chapter 6

Discussion and Limitations

6.1 Discussion

As the results show, the autoencoder network trained on the dog database, is able to reconstruct motions with exceptional accuracy. The generator model we use for data augmentation successfully synthesizes high quality & believable animations for the dog skeleton, and reasonably satisfactory animations for the human skeleton. Our ability to realize our proposed cross-domain motion retargeting framework, heavily relies on the capacity to accurately reconstruct motions, and augment the limited-in-size existing animal datasets. Therefore, concerning the matter of attaining a powerful core upon which we can build our framework, we have achieved our goal.

Utilizing an implementation of our second proposed method (Latent Space Transfer) we managed to attain results - albeit limited - which corroborate the idea of leveraging a reduced representation of motion (primal skeleton) in order to achieve cross-domain motion retargeting. Our FNN however, struggles to produce accurate latent space mappings for unseen data which are significantly different in nature to the training data, as a result of the

limited variety and restricted size of the training dataset. Additionally, the architecture chosen for the FNN might not be optimal for the task of Latent Space Transferring.

On the same matter of training our FNN, several attempts were made to train an FNN in the direction of human-to-dog, but to no avail. This is likely because of the difference in input-output dimensions for this particular direction. The input layer consists of only 448 neurons, while the output layer is made up of 512 neurons, thus the network has trouble converging. In the opposite direction, which is the one we provide results for, this is not a problem as the input layer is larger than the output layer.

Although this research has the broad scope of providing a scalable framework that successfully facilitates cross-domain motion retargeting, we have only managed to implement one of the two of our proposed methods for achieving our target, and attained limited results. This is mainly due to encountering different hurdles during the carrying out of this thesis, which imposed significant delays in the progression of the implementation. Most notably, when attempting to implement the Shared Latent Space method, the SparsePoser autoencoder had difficulties converging as a result of the 15% increase in network parameters caused by adding a single dummy joint. Despite several attempts to resolve this issue we did not find success. This combined with the fact that training the autoencoder itself takes tens of hours, stalled the implementation process considerably.

6.2 Limitations

Despite our framework's promising results, a plethora of problems is still to be addressed. Firstly, the complex structure of the SparsePoser generator combined with the relatively small datasets we have gathered for animals (excluding the human and dog domains), makes training the autoencoder to a satisfactory degree difficult. Secondly, generative methods are known to produce artifacts in synthesized data, which might prove to be detrimental to our goal, since we are heavily relying on data augmentation by generation. More specifically, because of the scarcity of data for certain domains, we are depending on generative

techniques for obtaining sufficient corresponding motion pairs to be used as training examples for our framework. In addition, the fact that we rely on simple repetitive motions for the construction of corresponding motion pairs means that our models are likely to produce limited results for complex inputs, or inputs drastically different to the training data. Finally, regarding our second proposed method for facilitating motion retargeting (Latent Space Trasnfer), the FNNs responsible for the transfer of latent space from one domain to another, are supposed to address the issue of latent space dimensionality matching. However, as evident by our results, when attempting to train an FNN in the direction where the input layer is smaller than the output layer, convergence is difficult at best.

Chapter 7

Conclusion

7.1 Conclusion

Motivated by the increasing need of reliable methods to generate novel data across different domains, we commenced our venture into the world of Cross-Domain Motion Retargeting. We set out with the initial goal of attaining a solid core upon which a Motion Retargeting framework can be built. During the progress of this thesis, we successfully make use of recent developments in the field of computer animation, to construct the building blocks for our cross-domain retargeting framework, which include an autoencoder network able to accurately reconstruct motion, and a generative method by which we can effectively augment our dataset. We then proceed to give a comprehensive description of two novel scalable methods for actualizing Cross-Domain Motion Retargeting in our established foundation. While several recent works attempt to implement Motion Retargeting, they are limited in scope as in they don't attempt to venture too far off the constraints of a certain domain. In contrast, our methods look to achieve Motion Retargeting regardless of the difference in skeletal structure. Specifically, the two approaches we propose are the Shared Latent Space

and Latent Space Transfer methods. The first method, imposes latent space dimensionality matching between a given domain and the human domain, with the goal of learning to minimize latent space loss with the human domain so as to allow for an all-to-all connectivity between supported species. This is achieved by introducing a novel latent space loss function in the autoencoder network, and training on corresponding motion sequence pairs. We address velocity & phase matching by providing two simple mutually exclusive solutions which consist of introducing two new loss functions into the autoencoder network. The second method, avoids the issue of latent space dimensionality matching all together, by providing an external independent Feedforward Neural Network which trains on corresponding motion pairs, effectively providing a latent space mapping function. In this thesis dissertation we have provided results for the second method, which prove the possibility of going from one latent space representation to another. Even though our results are limited, we are confident that addition of more diverse and extensive corresponding motion pairs, along with a deeper study into finding the optimal architecture, will yield much better results. All in all, we are optimistic about the future of this research and we firmly believe that through diligent work, the framework we advocate for will come to fruition.

7.2 Future Work

In the future, we intend to address the aforementioned limitations of our framework, as well as actually implement our first proposed method for motion retargeting (Shared Latent Space). Specifically, in order to get around the issue of non-convergence due to the dummy joint introduction, we plan to implement the second strategy for achieving latent space dimensionality matching - Sparse Joint Removal (section 4.4.1.3). Additionally, we will attempt to slightly reduce the complexity of the SparsePoser autoencoder network, so as to allow for convergence on smaller datasets, with the goal of incorporating more animals in the list of supported domains. We of course intend to diligently go through all mocap files in our possession, and extract & label all motion sequences which can be used as corresponding motion pairs. Different FNN architectures need to be tested in order to pinpoint the optimal design for the Latent Space Transfer mechanism, and finally we will utilize GANimator and potentially other generative methods to their fullest extent, taking advantage of features like conditional generation which allows for motion synthesis following a sampled root trajectory, in order to obtain more complex motion sequences to be used as corresponding motion pairs.

Currently, we do not provide any methods for minimizing error between source and target end-effector positioning. For instance, even in the scenario where our proposed methods for motion retargeting work perfectly, it's likely that they will underperform in cases where the input consists of an unusual unpredictable motion of a single limb / end-effector, e.g. flapping of an arm in an erratic manner. Assuming that all previous limitations are adequately addressed, and our goal of providing a reliable Cross-Domain Motion Retargeting framework is achieved, the next step would be to incorporate a mechanism by which end effector accuracy is controlled.

References

I. Aberman, K., Li, P., Lischinski, D., Sorkine-Hornung, O., Cohen-Or, D., & Chen, B. (2020). Skeleton-Aware Networks for Deep Motion Retargeting. *ACM Transactions on Graphics, Vol. 39, No. 4, Article 62*.

II. Adam Paszke, S. G. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS.* doi:https://doi.org/10.48550/arXiv.1912.01703

III. C Walker, C. J. (1998). Balance in the cat: role of the tail and effects of sacrocaudal transection. *Behav Brain Res.* doi:https://doi.org/10.1016/s0166-4328(97)00101-0

IV. He Zhang, S. S. (n.d.). Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics*, *37.* doi:https://doi.org/10.1145/3197517.3201366

V. Ilya Loshchilov, F. H. (2017). Decoupled Weight Decay Regularization. *ICLR 2019.* doi:https://doi.org/10.48550/arXiv.1711.05101

VI. Jose Luis Ponton, H. Y., Aristidou, A., & Carlos Andujar, N. P. (2023). SparsePoser: Real-time Full-body Motion Reconstruction from Sparse Date. *ACM Transactions on Graphics, Vol. 43, No. 1, Article 5*.

VII. Kramer, M. (1992). Autoassociative neural networks. *Computers & Chemical Engineering, 16*(4), 313-328. doi:https://doi.org/10.1016/0098-1354(92)80051-A

VIII. Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AiChE, 37*(2). doi:https://doi.org/10.1002/aic.690370209

IX. Nefeli Andreou, A. A. (2021). Pose Representations for Deep Skeletal Animation. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA'22.* doi:https://doi.org/10.48550/arXiv.2111.13907

X. Olaf Ronneberger, P. F. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *MICCAI 2015.* doi:https://doi.org/10.48550/arXiv.1505.04597

XI. Peizhuo Li, K. A.-H. (2022). GANimator: Neural Motion Synthesis from a Single Sequence. *SIGGRAPH 2022.* doi:https://doi.org/10.48550/arXiv.2205.02625

XII. Ponton, J. L. (n.d.). PyMotion. Retrieved from https://github.com/UPC-ViRVIG/pymotion

XIII. Qingqing Zhao, P. L.-H. (2023). Pose-to-Motion: Cross-Domain Motion Retargeting with Pose Prior. *arXiv:2310.20249* .

XIV. Reda, D., Won, J., Ye, Y., Panne, M. V., & Winkler, A. (2023). Physics-based Motion Retargeting from Sparse Inputs. *ACM Comput. Graph. Interact. Tech.*, *6.* doi:https://doi.org/10.48550/arXiv.2307.01938

XV. Ruben Villegas, J. Y. (2018). Neural Kinematic Networks for Unsupervised Motion Retargetting. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* doi:https://doi.org/10.48550/arXiv.1804.05653

XVI. Sigal Raab, I. L.-O. (2023). Single Motion Diffusion. *ICLR 2024 Spotlight.* doi:https://doi.org/10.48550/arXiv.2302.05905

XVII. University of Cyprus. (2013). Dance Motion Capture Database. Retrieved from https://dancedb.cs.ucy.ac.cy/