PMnet: Learning of Disentangled Pose and Movement for Unsupervised Motion Retargeting

Jongin Lim
ljin0429@snu.ac.kr
Hyung Jin Chang
h.j.chang@bham.ac.uk
Jin Young Choi
jychoi@snu.ac.kr

1 Department of ECE, ASRI, Seoul National University, Seoul, Korea
2 School of Computer Science, University of Birmingham, Birmingham, UK

Abstract

In this paper, we propose a deep learning framework for unsupervised motion retargeting. In contrast to the existing method, we decouple the motion retargeting process into two parts that explicitly learn poses and movements of a character. Here, the first part retargets the pose of the character at each frame, while the second part retargets the character’s overall movement. To realize these two processes, we develop a novel architecture referred to as the pose-movement network (PMnet), which separately learns frame-by-frame poses and overall movement. At each frame, to follow the pose of the input character, PMnet learns how to make the input pose first and then adjusts it to fit the target character’s kinematic configuration. To handle the overall movement, a normalizing process is introduced to make the overall movement invariant to the size of the character. Along with the normalizing process, PMnet regresses the overall movement to fit the target character. We then introduce a novel loss function that allows PMnet to properly retarget the poses and overall movement. The proposed method is verified via several self-comparisons and outperforms the state-of-the-art (sota) method by reducing the motion retargeting error (average joint position error) from 7.68 (sota) to 1.95 (ours).

1 Introduction

Motion retargeting is the process of applying a source motion to a new target character with different kinematic configurations. More formally, given a motion sequence of character $A$ and kinematic information of another character, $B$, the goal is to make $B$ imitate the motion of $A$ so that another motion sequence of $B$ can be generated. For this paper, we assumed that the skeletons have the same topology but different bone lengths and proportions. Due to its reusability for motion data, motion retargeting has been studied extensively in computer graphics [9, 11, 23, 34] and robotics [4, 6, 29, 33]. Most existing methods have formulated motion retargeting as a constrained optimization problem. However, since they usually rely on motion-specific constraints or manually designed kinematic constraints, these optimization-based methods cannot generalize to a wide range of motions and characters.

© 2019. The copyright of this document resides with its authors. It may be distributed unchanged freely in print or electronic forms.
Recent deep learning based methods [7, 10, 15, 16, 25, 26, 31] have succeeded in modeling human motion based on training using large-scale motion capture datasets [1, 17]. However, applying the deep learning framework to motion retargeting is challenging because there is a lack of paired motion data from different characters. Thus, a supervised learning scheme with the ground truth for the retargeted motion is impractical. Recently, Villegas et al. [36] proposed a recurrent neural architecture for motion retargeting. By exploiting cycle consistency [39], the possibility of unsupervised motion retargeting was presented. However, the resulting motions showed a large number of errors in preserving the pose of the input and also showed unrealistic motions, such as bodies floating above the ground.

In our study, to address the aforementioned limitations, we propose a novel architecture referred to as the pose-movement network (PMnet) for unsupervised motion retargeting, which learns frame-by-frame poses and overall movement separately. Here, the frame-by-frame pose is referred to as the relative coordinates from the root joint position, while the overall movement is referred to as the velocity of the root joint [15, 16]. In contrast to previous work [36], which learns the character motion in a recurrent architecture, we disentangle the character motion into frame-by-frame poses and overall movement to learn specialized and complementary representations from them.

The proposed method consists of two parts. The first part learns to ensure that the target character has a pose similar to the input character. To this end, we propose a novel architecture consisting of a pose encoder and two mapping networks. At each frame, the pose encoder encodes the skeleton invariant pose representation, and two mapping networks map the pose representation to unit quaternions for the target character to adapt to a different kinematic configuration than the input character. The second part learns to generate the overall movement of the target character, which makes the resulting motion seem realistic. To accomplish this, we propose a movement regressor network and a normalizing process that make the movement invariant to the size of the character. Then, to ensure that the aforementioned parts work properly, we design a novel training loss. Rather than using cycle consistency, our training loss consists of the following four loss terms: i) reconstruction, ii) perceptual pose, iii) motion discrimination, and iv) rotation constraint. By means of the new loss terms, PMnet implicitly learns to retarget motion while preserving the detailed pose of the input and making realistic movements. We validate the proposed method via several self-comparisons and show that it significantly outperforms the previous work, reducing the motion retargeting error from 7.68 to 1.95 in the same experimental setup.

2 Related Works

2.1 Motion Retargeting

To adapt the motion from source to target, Gleicher [11] formulated a spacetime constraint problem, then solved it numerically. Lee et al. [23] decoupled the motion retargeting problem, which solved the inverse kinematics problem for each frame and then adjusted multi-level B-spline curves for smooth results. Choi et al. [9] presented online motion retargeting technique by solving inverse kinematics problem under constraints and computing the changes in joint angles at each frame. Tak et al. [34] proposed per-frame motion retargeting framework which incorporates both kinematic constraints and dynamic constraints into Kalman filter formulation. Motion retargeting techniques can be applied to adapting motions from humans to humanoid robots [4, 6, 8, 28, 29, 30, 32]. While retargeting human mo-
tion onto humanoids additionally requires considerations, they are basically based on optimiza-
tions to meet specified constraints \[4\]. The aforementioned methods have limitations in
applying to large scale motion data since they relied on optimizations with motion-specific
constraints or manually designed kinematic constraints. Shon et al. \[33\] proposed Gaussian
Process model to learn common latent structure shared between sets of motion capture
data and corresponding poses from a humanoid robot, presenting robotic imitation of human
poses. But, this method requires a set of paired training data from two domains.

Recently, instead of numerical approaches, data-driven approaches based on deep learn-
ing framework have been proposed. Jang et al. \[19\] proposed a deep learning framework
that can produce 27 variations of the source motion with a set of different levels of arms,
legs, and torso lengths. However, this method requires fully-supervised training from the
 corresponding data pairs based on \[22\]. Furthermore, it requires post-optimization process
for the latent variable to meet the valid kinematic consistency. Villegas et al. \[36\] proposed
Neural Kinematic Network with forward kinematics layer and cycle consistency \[39\] based
objectives, which suggested the possibility of unsupervised motion retargeting. However, the
quality of resulting motion has still room for improvement.

2.2 Human Motion Modeling

Modeling human motion is a long standing problem in computer vision and machine learn-
ing. Early works used Restricted Boltzmann Machines \[35\], Markov Models \[24\] and Gauss-
ian Process \[13, 38\], but limited to small scale of motions. The most common strategy in
recent years is a data-driven approach where postures are reconstructed based on large-scale
motion capture datasets \[1, 17\]. Specifically, deep learning based methods have succeeded
in synthesizing or predicting plausible human motions \[7, 15, 16, 25\]. However, these meth-
ods can not be applied to motion retargeting because they require re-projection onto kine-
matic constraint to avoid invalid bone length configuration as they regress on joint positions.
Several works have used human poses described through 3D joint rotations \[10, 18, 26\],
which used the angle loss so that they were vulnerable to small angle errors at the root joint.
Pavllo et al. \[31\] represented rotations with unit quaternions and presented loss function
which performs forward kinematics on a skeleton to penalize position errors instead of angle
errors. However, these methods are not applicable when joint angles are not given and only
joint positions are observed.

3 Methodology

Let \(x^A_{1:T}\) be a source motion performed by the character \(A\), given as \(T\) frames of 3D joint
positions. Motion retargeting is the process of generating the motion \(x^B_{1:T}\) which is the same
action sequence performed by another character \(B\). Assuming that the kinematic configura-
tion of the target character is given as a T-pose skeleton, namely, \(ref_B\), Villegas et al. \[36\]
formulated motion retargeting problem as outputting 3D rotations for each joint parameter-
ized by unit quaternions and applying them to \(ref_B\). To this end, forward kinematics layer was
presented to compute the resulting joint positions from the joint rotations and \(ref_B\). Resulting
Neural Kinematic Network (NKN) with forward kinematics layer was trained based on cy-
cle consistency objectives and suggested the possibility of unsupervised motion retargeting.
However, NKN shows unsatisfactory results in preserving the details of input motions, and
it even shows unrealistic motions such as body floating or sinking when input motions have
dynamic movements. This is because cycle consistency is insufficient to capture the proper pose representation. Also, there is a lack of consideration for the overall movements of the character. Our research focuses on solving the aforementioned problems.

The motion capture data is often decomposed into two parts, the relative coordinates from the root joint position and the root joint’s velocities and direction (rotation with respect to the axis perpendicular to the ground) [15, 16]. In this paper, we viewed the former as frame-by-frame poses and the latter as a overall movement.

As shown in Figure 1, the input motion data $x^{A}_{1:T}$ is decomposed into frame-by-frame poses $p^{A}_{t}$ and the overall movement $v^{A}_{1:T}$ (see [16] for more details). Then, we decouple the process to handle the pose and the movement separately for the purpose of extracting different and complementary meanings. At each frame $t$, the proposed method outputs the unit quaternions $q^{B}_{t} \in \mathbb{R}^{4N}$, which represent the 3D rotations for each joint of the character $B$, from the input pose $p^{A}_{t} \in \mathbb{R}^{3N}$ of $A$ and the target T-pose skeleton of $B$, $ref_{B} \in \mathbb{R}^{3N}$, where $N$ is the number of joints. To make the target character $B$ follow the pose of the input character $A$ in good shape, we present a novel architecture that successfully encodes the skeleton invariant pose representation from the pose (see Section 3.1). The retargeted pose of the character $B$, $p^{B}_{t}$, is then computed by applying $q^{B}_{t}$ to $ref_{B}$ using forward kinematics (FK) layer. When processing the overall movement, on the other hand, the temporal context must be grasped. Therefore, $v^{B}_{1:T}$, the movement of the target character, is regressed from the entire input movement $v^{A}_{1:T}$ to capture the character’s overall movement. (see Section 3.2). Then, the retargeted motion $x^{B}_{1:T}$ can be computed by combining the obtained frame-by-frame pose and the overall movement of the character $B$. Note that learning proceeds in an unsupervised manner, i.e., there is no ground truth for the retargeted motion $x^{B}_{1:T}$. To this end, we further introduce a novel unsupervised learning scheme which allows the model to retarget motions while maintaining good pose and overall movement (see Section 3.3).

### 3.1 Architecture for Pose Retargeting

In this section, we present an architecture that produces the unit quaternions $q^{B}_{t}$ to make the target character $B$ follow the pose of the input $p^{A}_{t}$ at each frame. This is challenging because there is no ground truth for $p^{B}_{t}$, so we can not give a guide to what pose $B$ should take. The
The key insight to tackle this problem is to learn how to reconstruct $p_t^A$ first, then make minor modifications to fit with $B$.

Auto-encoders have shown reliable results for unsupervised feature learning \cite{14,37}. Similarly, we present the auto-encoding phase to reconstruct the input pose $p_t^A$. To encode the pose representation from $p_t^A$, we present a pose encoder $F_P$. That is,

$$\phi_t^A = F_P(p_t^A; W_P),$$

(1)

where $W_P$ denotes the connection weights of the pose encoder. The pose representation $\phi_t^A$ from the pose encoder conveys shape information about the pose at $t$ frame. Then, a mapping function $F_q$ is introduced to map the pose representation $\phi_t^A$ to the quaternion space. After the mapping, the output is normalized because it must be defined as unit quaternions to represent the 3D rotations as follows,

$$q_t^A = \frac{F_q(\phi_t^A; W_q)}{\|F_q(\phi_t^A; W_q)\|_2}.$$  

(2)

Here, $q_t^A$ represents the rotation to make the pose of $A$. Then, to make modifications to fit with $B$, another mapping function $F_\Delta$ is presented. $F_\Delta$ outputs the modifications on the quaternion space from the pose representation $\phi_t^A$ and the T-pose skeleton $ref_B$ as follows,

$$\Delta q = \frac{F_\Delta(\phi_t^A, ref_B; W_\Delta)}{\|F_\Delta(\phi_t^A, ref_B; W_\Delta)\|_2}.$$  

(3)

In the quaternion space, the product of two rotation quaternions $q_1$ and $q_2$, called the Hamilton product, represents the rotation equivalent to the rotation of $q_1$ followed by the rotation of $q_2$. That is, $q_t^B$, modified version of $q_t^A$ by $\Delta q$, can be expressed as a hamilton product of two quaternions. Then, the retargeted pose $p_t^B$ can be obtained as follows,

$$q_t^B = q_t^A \otimes \Delta q,$$

(4)

$$p_t^B = FK(q_t^B, ref_B).$$  

(5)

By learning to reconstruct $p_t^A$ by applying $q_t^A$ to $ref_A$ using FK layer, what rotations $q_t^A$ makes the desired pose can be learned. Thus, learning the subtle changes in $q_t^A$ is much easier than learning from a zero base (see Section 3.3).

### 3.2 Architecture for Movement Retargeting

In this section, we present an architecture which handles the overall movement. Unlike the pose, the temporal context must be grasped. To this end, we present a movement regressor $F_M$ which regresses the entire input movement $v_t^{A,T} \in \mathbb{R}^{4T}$. Recent studies suggest that convolutional models empirically outperform recurrent models \cite{5,20,27}. In our work, $F_M$ is realized by 1D convolutions on the time sequence.

Since the overall movement contains the $x$, $y$, $z$ velocities of the root joint, it is affected by the size of the character, e.g., for the same walking motion, the adult step is faster than the child step. To capture the scale invariant information about the overall movement, the
overall movement is normalized with respect to the scale of the character before fed into the movement regressor $F_M$. The scale factor $S$ of the character is defined as follows,

$$S = \frac{1}{N_s} \sum_{i \in H} d(i, \text{parents}(i)), \quad (6)$$

where $N_s$ be a scaling coefficient that adjusts $S$ not to be too large, $H$ be a set of joints whose elements are Root, Spine 0,1,2, Neck, LeftUpLeg, LeftLeg and LeftFoot, and $d(i, j)$ denotes the $L_2$ distance between $i$ and $j$. Then, normalizing is done by dividing the x,y,z velocities (except the rotation) by the corresponding scale factor. As shown in Figure 1, $v_{1:T}^A$ is normalized to $nv_{1:T}^A$ with respect to its scale factor $S_A$ then the movement regressor $F_M$ regresses the scale invariant overall movement $nv_{1:T}^B$ from $nv_{1:T}^A$. That is,

$$nv_{1:T}^B = F_M(nv_{1:T}^A; W_M). \quad (7)$$

Similarly, $nv_{1:T}^B$ is then denormalized to $v_{1:T}^B$ by multiplying the scale factor $S_B$ to the x, y, z velocities in $nv_{1:T}^B$.

### 3.3 Unsupervised Training for Motion Retargeting

In this section, we present the training loss which allows the aforementioned parts to work properly. Our loss consists of four parts:

**i) Reconstruction Loss.** First, the model learns to reconstruct $p_t^A$ by applying $q_t^A$ to $ref_B$ using FK layer. By minimizing the reconstruction error, $F_P$ learns to encode the meaningful pose representation $\phi_t^A$ while $F_q$ learns what rotations $q_t^A$ makes the desired pose given $\phi_t^A$. The reconstruction loss for the entire $T$ frames is defined as follows,

$$L_{\text{recon}} = \sum_t \| p_t^A - FK(q_t^A, ref_A) \|_2^2. \quad (8)$$

Additionally, meeting the end-effector positions may be critical to make the pose similar. In our work, joints that affect the end-effector positions have double the weight to the loss.

**ii) Perceptual Pose Loss.** By Eq. (8), the rotation $q_t^A$ makes the desired pose of $A$. $B$ should be the same pose of $A$ but minor modifications are needed because of the different kinematic configuration. To this end, we present the perceptual pose loss exploiting the pose encoder $F_P$. We feed the retargeted pose $p_t^B$ into $F_P$ then penalize the pose representation $\phi_t^B$ to be close to the pose representation of $A$, $\phi_t^A$. As learning proceeds to minimize the difference between the pose representations, the pose encoder $F_P$ learns to encode the skeleton invariant pose representation. Then, the modifications coming from the different kinematic configuration can be encoded in $\Delta q$. In our experiments, we minimize the $L_2$ distance between the pose representations and use $\lambda_p = 5$. That is,

$$L_{\text{pose}} = \lambda_p \| \phi_{1:T}^A - \phi_{1:T}^B \|_2^2. \quad (9)$$

**iii) Motion Discrimination Loss.** Now, we present the loss term that penalize to make the retargeted motion $x_{1:T}^B$ look realistic. We define the mean point trajectory $\bar{m}_{1:T}$ where $\bar{m}_t = \text{mean}(x_t) \in \mathbb{R}^3$, i.e., the mean position of all joints. Then, $\bar{m}_{1:T}^A$ and $\bar{m}_{1:T}^B$ are normalized with respect to its corresponding scale factors computed by Eq. (6). To avoid the unrealistic motion such as body floating or sinking, we make these normalized mean point trajectories be similar to each other. Exploiting the adversarial training [12], we present the motion
discriminator $D(\cdot;W_D)$ which discriminates the normalized mean point trajectories. Then, PMnet is trained to fool the discriminator as follows,

$$L_{\text{motion}} = \lambda_m \mathbb{E}_{x \sim p_B(x)} \left[ \log \left( 1 - D \left( \frac{\bar{m}^{B}_{1:T}}{S_B} \right) \right) \right],$$  \hspace{1cm} (10)

where $x \sim p_B(x)$ means the sampling from the distribution of retargeted motion generated by the proposed method and $\lambda_m = 2$ is used as a balancing parameter.

iv) Rotation Constraint Loss. The last term in our loss aims to limit the rotation angles to be within a certain range, which avoids the excessive bone twisting.

$$L_{\text{rot}} = \lambda_r \left( \| \max(0, |\text{euler}_y(q^A_{1:T})| - \alpha) \|^2_2 + \| \max(0, |\text{euler}_y(q^B_{1:T})| - \alpha) \|^2_2 \right).$$  \hspace{1cm} (11)

As in [36], $\text{euler}_y(\cdot)$ denotes the rotation angle around the plane parallel to the bone (i.e., y-axis). In our experiments, $\lambda_r = 10$ and $\alpha = 100^\circ$ are used.

Finally, the total loss is defined by the summation of all four loss terms as follows,

$$L_{\text{total}} = L_{\text{recon}} + L_{\text{pose}} + L_{\text{motion}} + L_{\text{rot}}.$$  \hspace{1cm} (12)

The training proceeds end-to-end, which updates the whole network parameters $W_P, W_q, W_\Delta,$ and $W_M$ and the discriminator parameter $W_D$ iteratively. As in [36], when constructing a mini batch, we chose the same character to the inputs with probability of $p = 0.5$ for training stability. We used the Adam optimizer [21] with a batch size of 16 and a learning rate of 0.0001 then trained the model for 15000 iteration.

3.4 Implementation

We used TensorFlow [3] for implementation. The pose encoder $F_P$ and two mapping networks $F_q$ and $F_\Delta$ are realized by fully-connected (fc) layers while the movement regressor $F_M$ and the discriminator $D$ are realized by 1D convolution (1DConv) layers. The detailed architectures are reported in Table 1. The source code is available at https://github.com/ljin0429/PMnet.

4 Experiments

We evaluated the motion retargeting capability of the proposed PMnet on the Adobe Mixamo dataset [2] and compared the results with the state-of-the-art method (NKN [36]). The NKN results were obtained using the code released by the authors. We also compared the results with the most trivial approach for motion retargeting which directly copies the input quaternions and velocities (Copy). For fair comparison, we followed the same experimental setup as described in NKN. The training set consisted of 1650 non-overlapping motion sequences for 7 characters. For training, we used randomly sampled 2-second clips (60 frames). For testing, we evaluated the methods on 185 scenarios. Each scenario involved retargeting the 4-second clip (120 frames) of the motion sequence. The ground truths of the testing sequences were also collected for quantitative evaluation. For quantitative evaluation, we evaluated a target character-normalized mean square error (MSE) on the joint positions throughout the entire sequences, which is the same metric presented in NKN. For qualitative evaluation, we visualized the results by rendering the animated 3D characters. Further, we validated the proposed method through several self-comparisons.
Table 1: Detailed architecture for $\mathcal{F}_P$, $\mathcal{F}_q$, $\mathcal{F}_\Delta$, $\mathcal{F}_M$ and $D$. We used the leak rate of 0.2 for leaky ReLU and applied dropout for the whole network with the keep probability of 0.8. Please visit https://github.com/ljin0429/PMnet for more details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Layer</th>
<th>Number of Neurons</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F}_P$</td>
<td>input</td>
<td>66</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>fc</td>
<td>512</td>
<td>ReLU</td>
</tr>
<tr>
<td></td>
<td>fc</td>
<td>512</td>
<td>ReLU</td>
</tr>
<tr>
<td></td>
<td>fc</td>
<td>512</td>
<td>ReLU</td>
</tr>
<tr>
<td></td>
<td>fc</td>
<td>512</td>
<td>ReLU</td>
</tr>
<tr>
<td>$\mathcal{F}_q$</td>
<td>input</td>
<td>512</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>fc</td>
<td>88</td>
<td>-</td>
</tr>
<tr>
<td>$\mathcal{F}_\Delta$</td>
<td>input</td>
<td>512</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>fc</td>
<td>88</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Layer</th>
<th>Kernel</th>
<th>Stride</th>
<th>Channel Depth</th>
<th>Activation</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F}_M$</td>
<td>input</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>60</td>
</tr>
<tr>
<td>1DConv</td>
<td>3</td>
<td>1</td>
<td>128</td>
<td>leaky ReLU</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>1DConv</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>60</td>
</tr>
<tr>
<td>1DConv</td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>leaky ReLU</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>1DConv</td>
<td>4</td>
<td>2</td>
<td>32</td>
<td>leaky ReLU</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>1DConv</td>
<td>4</td>
<td>2</td>
<td>64</td>
<td>leaky ReLU</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1DConv</td>
<td>4</td>
<td>2</td>
<td>128</td>
<td>leaky ReLU</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1DConv</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>sigmoid</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

4.1 Self-comparisons

In order to validate the effectiveness of the proposed method, we compared the PMnet with the following four variants: $wPP$-PMnet, $wM$-PMnet, $wR$-PMnet, and $eqR$-PMnet. In $wPP$-PMnet, $L_{\text{pose}}$ was omitted while, in $wM$-PMnet, $L_{\text{motion}}$ was omitted. Then, we investigated the effectiveness of reconstruction. We omitted $L_{\text{recon}}$ in $wR$-PMnet and used the equal penalty for all joints when computing $L_{\text{recon}}$ in $eqR$-PMnet.

As shown in Table 2, the four variants of PMnet also achieved higher performances than NKN and the Copy baseline. This demonstrates that the proposed PMnet architecture is effective for learning about the pose and movement for motion retargeting. The results of $wPP$-PMnet showed increased error compared with PMnet, reflecting that the perceptual pose loss $L_{\text{pose}}$ contributes to making the proper modifications to fit with the target character. Without $L_{\text{motion}}$ in $wM$-PMnet, there is no guide for retargeting the overall movement properly, which results in increased motion retargeting error. Without $L_{\text{recon}}$, as expected, $wR$-PMnet shows significant performance degradation. This means that the proposed scheme, which learns to reconstruct $A$ first and then modify it rather than directly learning $B$ from a zero-base, is effective. In addition, we confirmed that more weights to the joints that affect the end-effectors when computing $L_{\text{recon}}$ results in a slight improvement to the performance as compared with $eqR$-PMnet.
Figure 2: Qualitative comparisons. We present two motion retargeting results, from Mutant to Liam (left) and from Big Vegas to Malcolm (right). The number in orange on the upper left indicates the frame number.

4.2 Qualitative and Quantitative Comparisons

We confirmed that the proposed PMnet is capable of retargeting motion and significantly outperforms NKN, as shown in Table 2. For the NKN results, we listed both the performance we achieved using the provided code and the performance reported in their paper. PMnet showed superior performance (1.95) to all the others, reducing the error by 74.6% compared with NKN (7.68) and 78.3% compared with the Copy baseline (9.00).

Figure 2 shows the qualitative results of PMnet and NKN. We present the results from two testing sequences. The number in orange on the upper left indicates the frame number. As shown in the left sequence, unlike NKN, PMnet preserved the details of the input motion well, including the motion of the legs and tilt of the body. This demonstrates that our model is capable of encoding good pose representation and making proper modifications to adapt it to different kinematic configurations. Further, NKN showed unrealistic movements (floating body) as can be seen in the right sequence. Even in this case, PMnet also showed reliable results, which demonstrates the effectiveness of the proposed movement regressor with normalizing and denormalizing processes.

Figure 3 shows the overall movement from PMnet, NKN, and the ground truth where (a)-(c) depict the x, y, z velocities of the root joint and (d) depicts the rotation of the root joint. The overall movement of PMnet is much more similar to the ground truth, suggesting that the resulting motion from PMnet looks more realistic, as shown in Figure 2.
Table 2: Quantitative results on Mixamo dataset [2]. For the NKN results, the numbers on the left are the results of our experiments using the code provided from the authors, and the numbers in brackets indicate the performance reported in their paper.

<table>
<thead>
<tr>
<th>Method</th>
<th>MSE</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NKN (CVPR’18) [36]</td>
<td><strong>7.68 (8.51)</strong></td>
<td>Auto-Encoder_Cycle loss</td>
</tr>
<tr>
<td></td>
<td>10.50 (10.25)</td>
<td>Cycle + Adv loss</td>
</tr>
<tr>
<td>Copy</td>
<td>9.00</td>
<td>Copy input quaternions and velocities</td>
</tr>
<tr>
<td>wPP-PMnet</td>
<td>2.54</td>
<td>without $L_{\text{pose}}$</td>
</tr>
<tr>
<td>wM-PMnet</td>
<td>2.79</td>
<td>without $L_{\text{motion}}$</td>
</tr>
<tr>
<td>wR-PMnet</td>
<td>2.99</td>
<td>without $L_{\text{recon}}$</td>
</tr>
<tr>
<td>eqR-PMnet</td>
<td>2.15</td>
<td>equal weights for $L_{\text{recon}}$</td>
</tr>
<tr>
<td>PMnet</td>
<td><strong>1.95</strong></td>
<td>Ours</td>
</tr>
</tbody>
</table>

Figure 3: Plots for the overall movement where the horizontal axis represents the frame sequence and the vertical axis represents the corresponding velocity and rotation values. We present the larger plots at [https://github.com/ljin0429/PMnet](https://github.com/ljin0429/PMnet).

5 Conclusion

In this paper, we proposed a new unsupervised motion retargeting framework based on learning disentangled meanings of pose and movement. Our contributions can be summarized as follows: 1) We decoupled motion retargeting process into two parts, where the first part generates the pose of the target character in each frame and the second part regresses the overall movement. 2) To generate the target pose, a novel architecture for obtaining quaternions of the input and modifying it using the Hamilton product was proposed. 3) The movement regressor along with normalizing and denormalizing processes was proposed to generate realistic movements. 4) A new training loss was designed to afford our deep network the capability to generate a target motion that mimics the input pose well and shows realistic movements. The experiments show compelling results, illustrating that the proposed method significantly outperforms the state-of-the-art method qualitatively and quantitatively.

Acknowledgement

This work was supported by Next-Generation ICD program through NRF funded by Ministry of S&ICT [2017M3C4A7077582], ICT R&D program of MSIP/IITP [No.B0101-15-0552, Predictive Visual Intelligence Technology] and Brain Korea 21 Plus Project.
References

[1] Cmu graphics lab motion capture database. URL http://mocap.cs.cmu.edu/.


