Robust Solving of Optical Motion Capture Data by Denoising

DANIEL HOLDEN, Ubisoft La Forge, Ubisoft, Canada

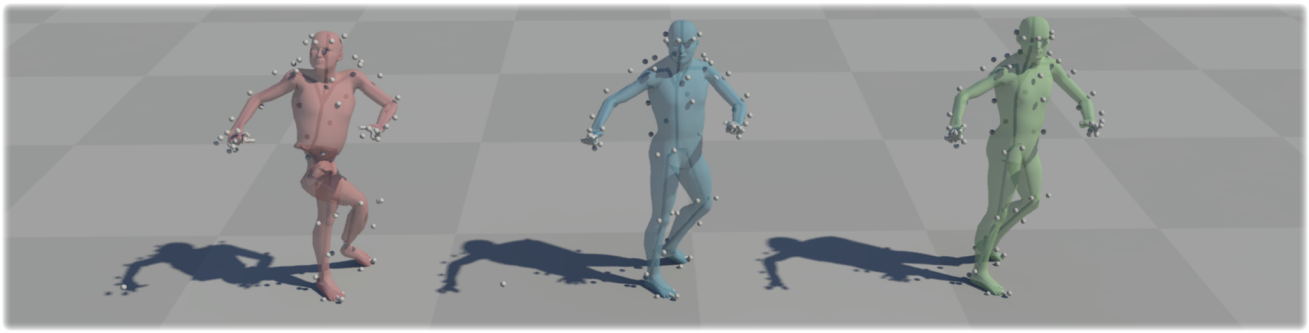

Fig. 1. Results of our method applied to raw motion capture data. Left: Raw uncleaned data. Middle: Our method. Right: Hand cleaned data.

Raw optical motion capture data often includes errors such as occluded markers, mislabeled markers, and high frequency noise or jitter. Typically these errors must be fixed by hand - an extremely time-consuming and tedious task. Due to this, there is a large demand for tools or techniques which can alleviate this burden. In this research we present a tool that sidesteps this problem, and produces joint transforms directly from raw marker data (a task commonly called "solving") in a way that is extremely robust to errors in the input data using the machine learning technique of *denoising*. Starting with a set of marker configurations, and a large database of skeletal motion data such as the CMU motion capture database [CMU 2013b], we synthetically reconstruct marker locations using linear blend skinning and apply a unique *noise function* for corrupting this marker data - randomly removing and shifting markers to dynamically produce billions of examples of poses with errors similar to those found in real motion capture data. We then train a deep denoising feed-forward neural network to learn a mapping from this corrupted marker data to the corresponding transforms of the joints. Once trained, our neural network can be used as a replacement for the solving part of the motion capture pipeline, and, as it is very robust to errors, it completely removes the need for any manual clean-up of data. Our system is accurate enough to be used in production, generally achieving precision to within a few millimeters, while additionally being extremely fast to compute with low memory requirements.

CCS Concepts: • **Computing methodologies** → **Motion capture**;

Additional Key Words and Phrases: Optical Motion Capture, Motion Capture Cleanup, Motion Capture, Machine Learning, Neural Networks, Denoising

Author's address: Daniel Holden, Ubisoft La Forge, Ubisoft, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, daniel.holden@ubisoft.com.

0730-0301/2018/3-ART165 $15.00
https://doi.org/0000001.0000001_2

ACM Reference Format:
Daniel Holden. 2018. Robust Solving of Optical Motion Capture Data by Denoising. *ACM Trans. Graph.* 38, 1, Article 165 (March 2018), 12 pages. https://doi.org/0000001.0000001_2

1 INTRODUCTION

Although a large number of different motion capture systems have been developed using many kinds of technologies, optical motion capture still remains the main technique used by most large movie and game studios due to its high accuracy, incredible flexibility, and comfort of use. Yet, optical motion capture has one major downside which severely limits the throughput of data that can be processed by an optical motion capture studio - namely that after each shoot the raw marker data requires "cleaning", a manual process whereby a technician must fix by hand all of the errors found in the data such as occluded markers, mislabeled markers, and high frequency noise. Although this process can often be accelerated by commercial software which can provide powerful tools for aiding in the cleanup of motion capture data [Vicon 2018], it can still often take several hours per capture and is almost always the most expensive and time consuming part of the pipeline. Once the data has been cleaned, further automatic stages are required including "solving", where rigid bodies are fitted to groups of markers, and "retargeting", where inverse kinematics is used to recover the local joint transformations for a character with a given skeleton topology [Xiao et al. 2008].

In this paper we propose a data-driven approach to replace the solving stage of the optical motion capture pipeline. We train a deep denoising feed-forward neural network to map from marker positions to joint transforms directly. Our network is trained using a custom noise function designed to emulate typical errors that may appear in marker data including occluded markers, mislabeled markers, and marker noise. Unlike conventional methods which fit rigid bodies to subsets of markers, our technique is extremely robust to errors in the input data, which completely removes the need for any manual cleaning of marker data. This results in a motion capture pipeline which is completely automatic and can therefore achieve a much higher throughput of data than existing systems.

To train our system, we require a large database of skeletal motion capture data such as the CMU motion capture database and a set of marker configurations, specified by the local marker offsets from the joints and their associated skinning weights. From this we synthetically reconstruct marker locations in the database using linear blend skinning, and then emulate the kinds of errors found in motion capture data using a custom noise function. This corrupted motion capture data is used to train the neural network.

To evaluate our technique, we present the results of our method on two different skeleton configurations and marker sets, and in a number of difficult situations that previously required extensive manual cleaning. To validate our design choices we compare our neural network architecture and training procedure against several alternative structures and show that our design decisions perform best.

The core contribution of this research is a production-ready optical motion capture solving algorithm which is fast to compute, has low memory requirements, achieves high precision, and completely removes the need for manual cleanup of marker data.

2 RELATED WORK

In this section we first review previous approaches tackling the problem of motion capture cleanup, followed by previous works which use the machine learning concept of *denoising* to solve problems in computer graphics.

Motion Capture Cleanup. To tackle the problem of motion capture cleanup, most researchers have looked towards techniques which introduce some *prior belief* about the behavior of motion capture markers or skeletal joints. There are two significant kinds of *priors* that researchers have used to fix broken data: temporal priors [Aristidou and Lasenby 2013; Baumann et al. 2011; Burke and Lasenby 2016; Dorfmüller-Ulhaas 2005; Liu et al. 2014; Ringer and Lasenby 2002; Zordan and Van Der Horst 2003], which exploit the fact that markers and joints must follow the laws of physics in their movement and cannot, for example, jump instantaneously to new locations - and pose-based priors [Chai and Hodgins 2005; Feng et al. 2014b; Li et al. 2010; Sorkine-Hornung et al. 2005; Tautges et al. 2011], which encode some knowledge about which poses may be possible for the character to achieve and which are unlikely.

An interesting physically based prior is introduced by Zordan et al. [2003] who present a novel solution to the solving and retargeting parts of the motion capture pipeline, using a physically based character which tracks the motion capture data. Virtual springs are attached between the markers and a physical character model, and resistive joint torques applied to force the character follow the marker data with minimal force. The proposed system sidesteps the retargeting stage of the pipeline while additionally allowing the character to avoid a number of unrealistic poses, but does not perform well in the case of marker swaps and is slow to compute as it requires a physical simulation to be performed.

Another approach that can be considered primarily a prior over the dynamic behavior of markers is data-driven marker gap filling [Baumann et al. 2011; Feng et al. 2014a; Liu and McMillan 2006]. In these approaches a large database of marker data is used to retrieve motion data capable of filling a given gap, which is then further edited to produce the in-between marker motion. A similar joint-space approach is presented by Aristidou et al. [2018] who use the self-similarity present in motions to avoid the use of an external database. Like ours, these approaches are data-driven and so can generate high quality results when the right kind of data is present, yet unlike ours, they tend to fail with more complicated errors such as marker swaps.

A popular PCA pose-based prior is introduced by Chai et al. [2005] and applied to sparse accelerometer data by Tautges et al. [2011]. Using a large database of motion capture data and local PCA a performance capture system is produced that takes as input low dimensional signals consisting of just a few marker positions and produces as output the full character motion. The PCA-decomposition produces a manifold in the pose-space which can additionally be used as a prior for removing errors in the input signal. Akhter et al. [2012] extend similar linear models by factorizing them into temporal and spatial components. Since PCA and other similar basis models are linear techniques it is hard to predict how they may behave under complex errors such as marker swaps, while using local linear models does not scale to large amounts of data as it requires many local basis to be constructed and interpolated.

Another pose-based prior is BoLeRO [Li et al. 2010], which uses a prior belief about the distances between markers and the joint lengths to fill gaps when markers are occluded. Additional soft constraints about the dynamics can be added and an optimization problem solved to fill gaps where markers are occluded. While this approach produces very accurate results for marker occlusions, it cannot deal well with marker swaps.

Our approach is a data-driven pose-based prior, and so it broadly has the same advantages and disadvantages of existing data-driven techniques. The use of a neural network achieves greater scalability and performance over techniques such as k-nearest neighbor which scales poorly with the size of the training data. It also allows us to deal robustly with marker swaps, not just marker occlusions - something that is rarely covered by existing work. Since we do not design our algorithm explicitly around the types of error we wish to fix, instead encoding this aspect of the research using a custom noise function, we retain a much greater level of flexibility and adaptability.

Denoising Neural Networks. The machine learning concept of *denoising* has been used to achieve state of the art results on a large number of problems in computer graphics.

Classically, denoising has been used as an unsupervised learning technique to train autoencoders to produce dis-entangled representations of data [Vincent et al. 2010]. Such autoencoders are naturally suited for tasks such as data recovery or in-painting [Xie et al. 2012; Yeh et al. 2017], but have also been used to improve classification or regression performance due to their ability to learn natural representations [II et al. 2015; Varghese et al. 2016]. Another popular form of denoising autoencoder is the Deep Boltzmann Machine [Salakhutdinov and Hinton 2009] which learns a compression of the input by stochastically sampling binary hidden units and attempting to reconstruct the input data. In this case the noise is represented by the stochastic sampling process and acts as a regularizer, ensuring

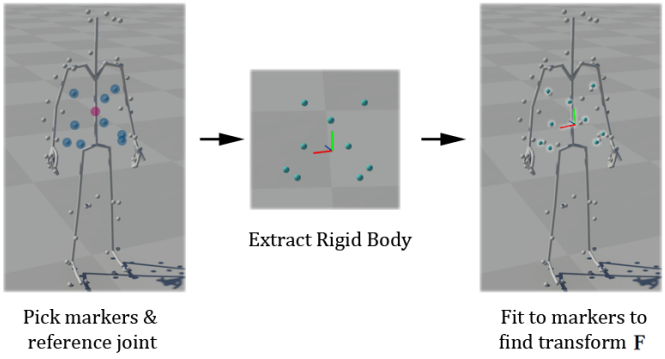

Fig. 2. Our rigid body fitting procedure. First, a subset of markers and a reference joint are chosen. Second, a rigid body is extracted using the average position of these makers in the database relative to the chosen reference joint. Finally, this rigid body is fitted back into the data using rigid body alignment.

the neural network learns to only output values found within the domain of the training data distribution [Hu et al. 2016].

More recently, neural networks have been used for the task of denoising Monte-Carlo renderings [Bako et al. 2017; Chaitanya et al. 2017]. Like our research, these methods benefit from synthetically producing corrupt data using a custom noise function, for example removing subsections of an image or adding Gaussian noise to pixels.

Most similar to our research are several existing works which use deep neural networks to fix corrupted animation data by autoencoding [Holden et al. 2016, 2015; Mall et al. 2017; Taylor et al. 2007; Xiao et al. 2015]. Although effective, these techniques primarily work in the joint space of the character and in some cases (E.G. that shown in Fig 1) this representation makes it extremely difficult to reconstruct the animation to a high enough accuracy without information from the original marker locations.

We take inspiration from these existing works using denoising, and apply a similar approach to the solving part of the motion capture pipeline.

3 PRE-PROCESSING

Our method starts with two inputs: a large database of skeletal motion capture data, and a set of marker configurations from a variety of different capture subjects.

Scaling. Before training, and during runtime, we scale all motion data such that the character has a uniform height. This form of normalization ensures we don't have to explicitly deal with characters of different heights in our framework, only characters of different proportions. An appropriate scaling factor can either be computed from the T-pose using the average length of the character's joints, or extracted directly from the motion capture software where it is often calculated during the calibration stage. Other than this, our system makes no assumptions about the capture subject and is explicitly designed to work well on a variety of subjects of different body shapes and proportions. All processes described from now on are therefore assumed to take place after scaling.

Representation. Given a character of j joints, and a data-set of n poses, we represent animation data using the joint's global homogeneous transformation matrices using just the rotation and translation components $\mathbf{Y} \in \mathbb{R}^{n \times j \times 3 \times 4}$. Given m markers, we represent the local marker configurations for each of the n different poses in the database using the local offset from each marker to each joint $\mathbf{Z} \in \mathbb{R}^{n \times m \times j \times 3}$, and the skinning weights associated with these marker offsets $\mathbf{w} \in \mathbb{R}^{m \times j}$. While technically we support local offsets that are time varying, in practice they almost always remain constant across motions of the same actor. Similarly, for simplicity we consider the skinning weights constant across all marker configurations. For joints which are not assigned to a given marker we set the skinning weight and offset to zero.

Linear Blend Skinning. Using this representation we can compute a set of *global* marker positions $\mathbf{X} \in \mathbb{R}^{n \times m \times 3}$ using the linear blend skinning function $\mathbf{X} = \text{LBS}(\mathbf{Y}, \mathbf{Z})$, which can be defined as follows:

$$\text{LBS}(\mathbf{Y}, \mathbf{Z}) = \sum_{i=0}^{j} \mathbf{w}_i \odot (\mathbf{Y}_i \otimes \mathbf{Z}_i). \tag{1}$$

Here we compute a sum over j joints, where the $\otimes$ function represents the homogeneous transformation matrix multiplication of each of the n marker offsets in $\mathbf{Z}_i$ by the n transformation matrices $\mathbf{Y}_i$, computed for each of the m markers, and the $\odot$ function represents a component-wise multiplication, which weights the contribution of each of the j joints in the resulting transformed marker positions, computed for each of the n poses.

Although we use linear blend skinning, any alternative skinning function such as Dual Quaternion Skinning [Kavan et al. 2008] should also be suitable, and more accurate skinning methods may produce even improved results.

Local Reference Frame. For data-driven techniques it is important to represent the character in some local reference frame. To robustly find a local reference frame to describe our data which does not involve knowing the joint transforms ahead of time, we make use of rigid body alignment [Besl and McKay 1992]. From a subset of chosen markers around the torso we calculate the mean location in the database of these markers relative to a chosen joint (usually one of the spine joints) and use these as the vertices of a rigid body we then fit into the data. After performing this process for all n given poses, the result is a set of n reference frames $\mathbf{F} \in \mathbb{R}^{n \times 3 \times 4}$ from which we can describe the data locally. For a visual description of this process please see Fig 2.

Statistics. After computing a set of local reference frames $\mathbf{F}$ and transforming every pose in our database into the local space, we calculate some statistics to be used later during training. First, we calculate the mean and standard deviation of our joint transformations $\mathbf{y}^{\mu} \in \mathbb{R}^{j \times 3 \times 4}$, $\mathbf{y}^{\sigma} \in \mathbb{R}^{j \times 3 \times 4}$, followed by the mean and covariance of the marker configurations $\mathbf{z}^{\mu} \in \mathbb{R}^{m \times j \times 3}$, $\mathbf{z}^{\Sigma} \in \mathbb{R}^{(m \times j \times 3) \times (m \times j \times 3)}$ respectively. We then compute the mean and standard deviation of marker locations $\mathbf{x}^{\mu} \in \mathbb{R}^{m \times 3}$, $\mathbf{x}^{\sigma} \in \mathbb{R}^{m \times 3}$ (where $\mathbf{X}$ can be computed using LBS). Finally we compute what we call the *pre-weighted*

ALGORITHM 1: This algorithm represents a single iteration of training for our network. It takes as input a mini-batch of n poses $\mathbf{Y}$ and updates the network weights θ.

Function *Train* ($\mathbf{Y} \in \mathbb{R}^{n \times j \times 3 \times 4}$, $\mathbf{F} \in \mathbb{R}^{n \times 3 \times 4}$)
  Transform joint transforms into local space.
  $\mathbf{Y} \leftarrow \mathbf{F}^{-1} \otimes \mathbf{Y}$
  Sample a set of marker configurations.
  $\mathbf{Z} \sim \mathcal{N}(\mathbf{z}^{\mu}, \mathbf{z}^{\Sigma})$
  Compute global marker positions via linear blend skinning.
  $\mathbf{X} \leftarrow \text{LBS}(\mathbf{Y}, \mathbf{Z})$
  Corrupt markers.
  $\hat{\mathbf{X}} \leftarrow \text{Corrupt}(\mathbf{X})$
  Compute pre-weighted marker offsets.
  $\hat{\mathbf{Z}} \leftarrow \sum_{i=0}^{j} \mathbf{w}_i \odot \mathbf{Z}_i$
  Normalize data, concatenate, and input into neural network.
  $\hat{\mathbf{X}} \leftarrow (\hat{\mathbf{X}} - \mathbf{x}^{\mu}) \div \mathbf{x}^{\sigma}$
  $\hat{\mathbf{Z}} \leftarrow (\hat{\mathbf{Z}} - \hat{\mathbf{z}}^{\mu}) \div \hat{\mathbf{z}}^{\sigma}$
  $\hat{\mathbf{Y}} \leftarrow \text{NeuralNetwork}(\left[\hat{\mathbf{X}}\, \hat{\mathbf{Z}}\right]; \theta)$
  Denormalize, calculate loss, and update network parameters.
  $\hat{\mathbf{Y}} \leftarrow (\hat{\mathbf{Y}} \odot \mathbf{y}^{\sigma}) + \mathbf{y}^{\mu}$
  $\mathcal{L} \leftarrow |\lambda \odot (\hat{\mathbf{Y}} - \mathbf{Y})|_1 + \gamma \|\theta\|_2^2$
  $\theta \leftarrow \text{AmsGrad}(\theta, \nabla \mathcal{L})$
End

ALGORITHM 2: Given marker data $\mathbf{X}$ as input this algorithm is used to randomly occlude markers (placing them at zero) or shift markers (adding some offset to their position).

Function *Corrupt* ($\mathbf{X} \in \mathbb{R}^{n \times m \times 3}$; $\sigma^o \in \mathbb{R}$, $\sigma^s \in \mathbb{R}$, $\beta \in \mathbb{R}$)
  Sample probability at which to occlude / shift markers.
  $\alpha^o \sim \mathcal{N}(\mathbf{0}, \sigma^o) \in \mathbb{R}^n$
  $\alpha^s \sim \mathcal{N}(\mathbf{0}, \sigma^s) \in \mathbb{R}^n$
  Sample using clipped probabilities if markers are occluded / shifted.
  $\mathbf{X}^o \sim \text{Bernoulli}(\min(|\alpha^o|, 2\sigma^o)) \in \mathbb{R}^{n \times m}$
  $\mathbf{X}^s \sim \text{Bernoulli}(\min(|\alpha^s|, 2\sigma^s)) \in \mathbb{R}^{n \times m}$
  Sample the magnitude by which to shift each marker.
  $\mathbf{X}^v \sim \text{Uniform}(-\beta, \beta) \in \mathbb{R}^{n \times m \times 3}$
  Move shifted markers and place occluded markers at zero.
  return $(\mathbf{X} + \mathbf{X}^s \odot \mathbf{X}^v) \odot (\mathbf{1} - \mathbf{X}^o)$
End

local offsets $\hat{\mathbf{Z}} \in \mathbb{R}^{n \times m \times 3}$, $\hat{\mathbf{Z}} = \sum_{i=0}^{j} \mathbf{w}_i \odot \mathbf{Z}_i$, along with their mean and standard deviation $\hat{\mathbf{z}}^{\sigma} \in \mathbb{R}^{m \times 3}$, $\hat{\mathbf{z}}^{\mu} \in \mathbb{R}^{m \times j \times 3}$. The pre-weighted local offsets are a set of additional values provided as an input to the neural network to help it distinguish between characters with different body proportions or marker placements. Providing the neural network with the full set of local offsets $\mathbf{Z}$ would be inefficient (each frame would require $m \times j \times 3$ values) so we instead compute the *pre-weighted* local offsets to act as a summary of the full set of local offsets in a way that only uses three values per-marker. For markers skinned to a single joint (often most markers) these pre-weighted offsets will contain exactly that offset to that single joint, while for markers skinned to multiple joints it will be a weighted sum of the different offsets. In this way the pre-weighted offsets contain most of the information about the marker layout but in a far more condensed form.

This concludes the pre-processing required by our method.

4 TRAINING

In this section we describe the neural network structure used to learn a mapping between marker locations and joint transforms and the procedure used to train it.

Our network works on a per-pose basis, taking as input a batch of n frames of marker positions $\mathbf{X}$ and the associated *pre-weighted* marker configurations $\hat{\mathbf{Z}}$, and producing as output a corresponding batch of n joint transforms $\mathbf{Y}$. The structure of our network is a six layer feed-forward ResNet [He et al. 2015] described in Fig 4. Each ResNet block uses 2048 hidden units, with a "ReLU" activation function and the "LeCun" weight initialization scheme [LeCun et al. 1998]. As we add noise to the input in the form of marker corruption, we find that no additional regularization in the form of Dropout [Srivastava et al. 2014] is required.

We now describe the training algorithm presented in more detail in Algorithm 1 and summarized in Fig 3. Given a mini-batch of n poses $\mathbf{Y}$, we first sample a batch of n marker configurations $\mathbf{Z}$ using the mean $\mathbf{z}^{\mu}$ and covariance $\mathbf{z}^{\Sigma}$ computed during preprocessing. From these we compute the marker positions $\mathbf{X}$ using linear blend skinning. We then corrupt these marker positions using the Corrupt function (described in Algorithm 2), producing $\hat{\mathbf{X}}$. We compute the *pre-weighted* marker offsets $\hat{\mathbf{Z}}$ to summarize the sampled marker configuration. We normalize the computed marker positions and summarized local offsets using the means and standard deviations $\mathbf{x}^{\mu}, \mathbf{x}^{\sigma}, \hat{\mathbf{z}}^{\mu}, \hat{\mathbf{z}}^{\sigma}$ computed during preprocessing. We then input these variables into the neural network to produce $\hat{\mathbf{Y}}$. We denormalize this using $\mathbf{y}^{\mu}$ and $\mathbf{y}^{\sigma}$, and compute a weighted l^1 norm of the difference between $\hat{\mathbf{Y}}$ and the joint transforms originally given as input $\mathbf{Y}$, scaled by a given set of user weights λ. Finally, we compute the gradient of this loss, along with the gradient of a small l^2 regularization loss where $\gamma = 0.01$, and use this to update the weights of the neural network using the AmsGrad algorithm [Reddi et al. 2018], a variation of the Adam [Kingma and Ba 2014] adaptive gradient descent algorithm.

This procedure is repeated with a mini-batch size of 256, until the training error converges or the validation error increases. We implement the whole training algorithm in Theano [Theano Development Team 2016] including the linear blend skinning and corruption functions. This allows us to evaluate these functions dynamically on the GPU at training time.

Our training function has a number of user parameters that must be set, the most important of which is the user weights λ. These weights must be tweaked to adjust the importances of different joint rotations and translations in the cost function while additionally accounting for the different scales of the two quantities. This weighting also allows for adjusting the importance when there is an imbalance in the distribution of joints around the character such as when the character has many finger joints. In our results we use two different constant factors for balancing the weighting of rotations and translations respectively, and then slightly scale up the weights of some important joints (such as the feet) and slightly scale down the weights for some other less important joints (such as the fingers).

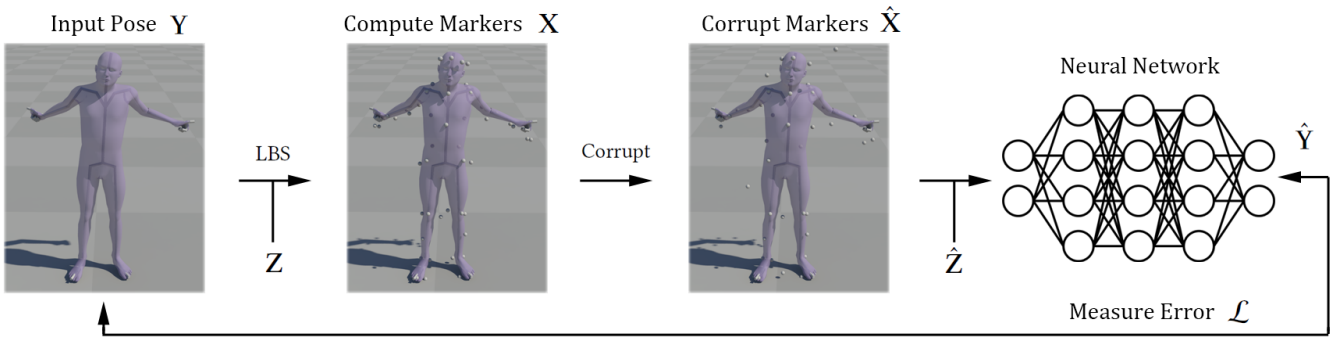

Fig. 3. Given an input pose Y we sample a marker configuration Z and reconstruct marker positions X. We then corrupt these marker positions to produce $\hat{\mathbf{X}}$, and input this, along with the *pre-weighted* marker offsets $\hat{\mathbf{Z}}$ into a neural network which produces joint transforms $\hat{\mathbf{Y}}$. We compare $\hat{\mathbf{Y}}$ to the original input pose Y, and use the error $\mathcal{L}$ to update the neural network weights.

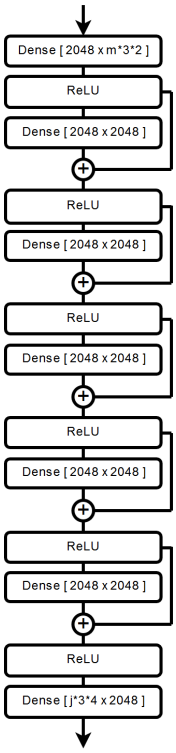

Fig. 4. Diagram of our network consisting of six layers of 2048 hidden units, using five residual blocks.

Our corruption function described in Algorithm 2 is designed to emulate marker occlusions, marker swaps, and marker noise. It does this by either removing markers or adjusting marker positions. Surprisingly, we found that randomly shifting a marker's position is a more effective corruption method than actually swapping marker locations in the data directly. One reason for this is that markers can sometimes swap with markers from other characters or objects in the environment which don't appear in the input. The parameters of this corruption function $\sigma^o, \sigma^s, \beta$ control the levels of corruption applied to the data. The value σ^o adjusts the probability of a marker being occluded, σ^s adjusts the probability of a marker being shifted out of place, and β controls the scale of the random translations applied to shifted markers. We set these values to 0.1, 0.1, and 50cm respectively, which we found struck a good balance between preserving the original animation and ensuring the network learned to fix corrupted data.

5 RUNTIME

After training, applying our method to new data is fairly straightforward and consists of the following stages. First, we use a simple procedure to remove outliers; markers which are located very far from related markers. Second, we fit the rigid body found during the preprocessing stage to the subset of non-occluded markers to find a local reference frame for the data. Next, we transform the marker positions into the local reference frame and pass them to the neural network to produce the resulting joint transforms, afterwards transforming them back in the global reference frame. Then, we post-process these transforms using a basic temporal prior in the form of a Savitzky-Golay filter [1964] which removes any residual high frequency noise or jitter from the output. Finally, we orthogonalize the rotational parts of the joint transforms and pass them to a custom Jacobian inverse-kinematics based retargeting solution to extract the local joint transforms. Each process is now described in more detail below.

Outlier Removal. We use a basic outlier removal technique which uses the distances between markers to find markers which are obviously badly positioned. If found, any "outlier" markers are considered occluded from then on. This simple algorithm is designed to be translation and rotation invariant as it runs before the rigid body fitting and tends to only occlude markers very obviously in the wrong position. It is only effective due to the robustness of the rest of our pipeline.

We start by computing the pairwise marker distances $\mathbf{D} \in \mathbb{R}^{n \times m \times m}$ for each frame in the training data X. From this we compute the minimum, maximum, and range of distances across all frames $\mathbf{D}_{min} \in \mathbb{R}^{m \times m}$, $\mathbf{D}_{max} \in \mathbb{R}^{m \times m}$, $\mathbf{D}_{range} = \mathbf{D}_{max} - \mathbf{D}_{min}$ (shown in Fig 5).

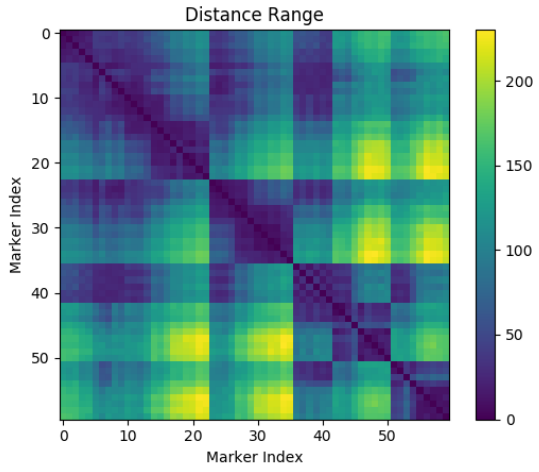

Fig. 5. Visualization of the distance range matrix $\mathbf{D}_{range}$. Sets of markers with a small range of distances represent more rigidly associated markers. The outlier removal process is more sensitive to violated distance constraints within these small ranges, yet allows for much larger distance variations for markers which have a larger range of distances present in the training data.

At runtime, given a new set of marker pairwise distances for every frame in the input $\hat{\mathbf{D}} \in \mathbb{R}^{n \times m \times m}$, we perform an iterative process for each frame which repeatedly removes (occludes) the marker which violates the following set of constraints to the greatest degree,

$$\hat{\mathbf{D}} < \mathbf{D}_{min} - \delta\, \mathbf{D}_{range},$$
$$\hat{\mathbf{D}} > \mathbf{D}_{max} + \delta\, \mathbf{D}_{range}.$$

Each time a marker is removed we also remove the marker's corresponding row and column from the distance matrices involved. Once all markers are within the range of these constraints we stop the process. This process effectively removes markers which are either much further from other markers than their largest recorded distance in the training data, or much nearer to other markers than their smallest recorded distance in the training data, with a sensitivity related to overall range of distances in the data, scaled by the user constant δ (set to 0.1 in this work).

Solving. Once outliers have been removed from the input we fit the rigid body extracted in the preprocessing stage and described in Section 3 to the markers of the body which are not occluded and use this to find a local reference frame $\mathbf{F}$. It should be noted that at least four markers associated with the rigid body must be present for this process to be successful [Besl and McKay 1992]. Next, we set the position of any occluded markers to zero to emulate the behavior of the corruption function described in Algorithm 2. Finally, we put the resulting marker positions and pre-weighted marker configurations through the neural network to produce joint transforms, which we convert back into the world space using the inverse reference frame $\mathbf{F}^{-1}$.

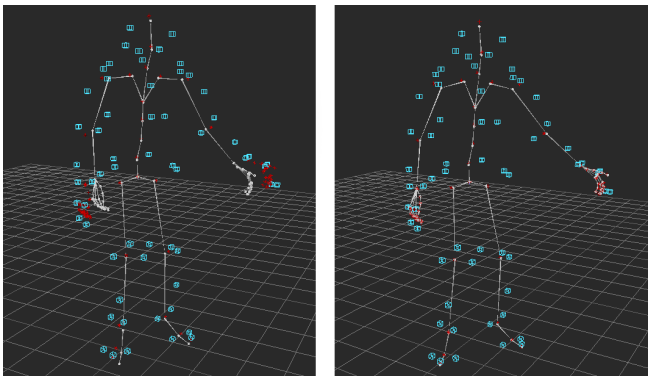

Fig. 6. Comparison showing our result before and after retargeting has been performed. Left: Shown in grey is the initial character state with local joint angles computed by mulitplying each joint's rotation in Y by the inverse of its parent's rotation. Shown in red are the target joint positions extracted from Y. Shown in blue are the original marker positions given as input. Right: Character state after retargeting has been performed. The joint positions now match those given in Y. The resulting local joint transforms are the final output of our method.

Character	Markers (m)	Joints (j)	Training Frames (n)
Production	60	68	5,397,643
Research	41	31	5,366,409

Table 1. Details of the characters used in our results and evaluation.

Filtering. As our method works on a per-pose basis, when a large number of markers appear or disappear from view quickly this can occasionally introduce jittery movements in the resulting joint transforms. To tackle this we introduce a basic post-process in the form of a Savitzky-Golay filter [1964]. We use a filter with polynomial degree three and window size of roughly one eighth of a second. This filter we apply to each component of the joint transformation matrices individually. Using a degree three polynomial assumes locally that the data has a jerk of zero, which in many cases is a good prior for motion capture data [Flash and Hogans 1985]. For motions which contain movements that break this assumption (such as those with hard impacts) it is possible for a user to tweak the window width of this filter to achieve a sharper, less smoothed out result. For more discussion about the advantages and limitations of this approach please see Section 8.

Retargeting. After filtering, we orthogonalize the rotational parts of these joint transforms using SVD (alternatively Gram-Schmidt orthogonalization can be used) and pass the results to a custom Jacobian Inverse Kinematics solver based on [Buss 2004] to extract the local joint transformations. Although we use this retargeting solution, we expect any standard retargeting solution to work equally well such as Autodesk's HumanIK [Autodesk 2016]. For a visual demonstration of how our results look before and after retargeting please see Fig 6.

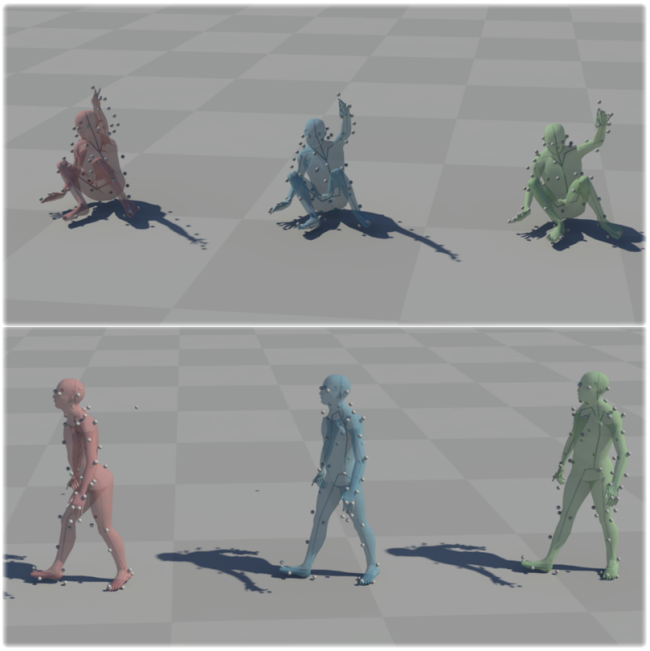

Fig. 7. Results of our method applied to raw motion data. Left: Raw uncleaned data. Middle: Our method. Right: Hand cleaned data.

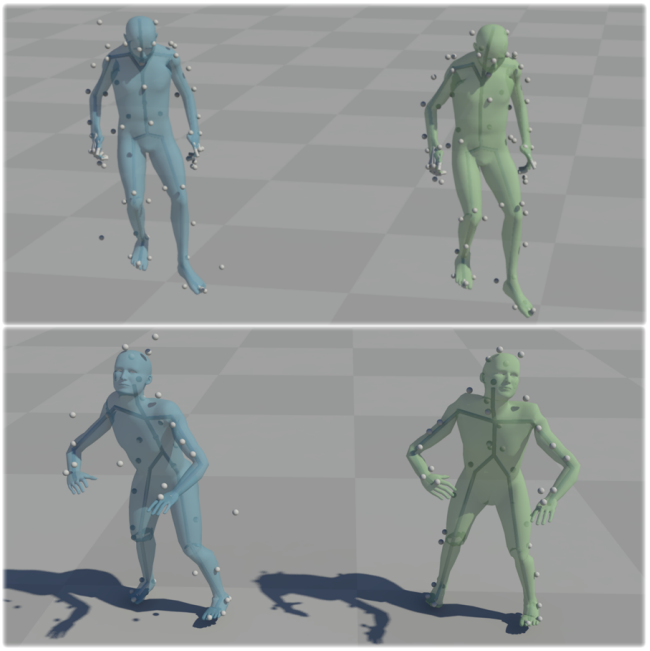

Fig. 8. Results of our method applied to motion corrupted using our custom noise function. Left: Our Method. Right: Ground Truth. Top: Production Character. Bottom: Research Character.

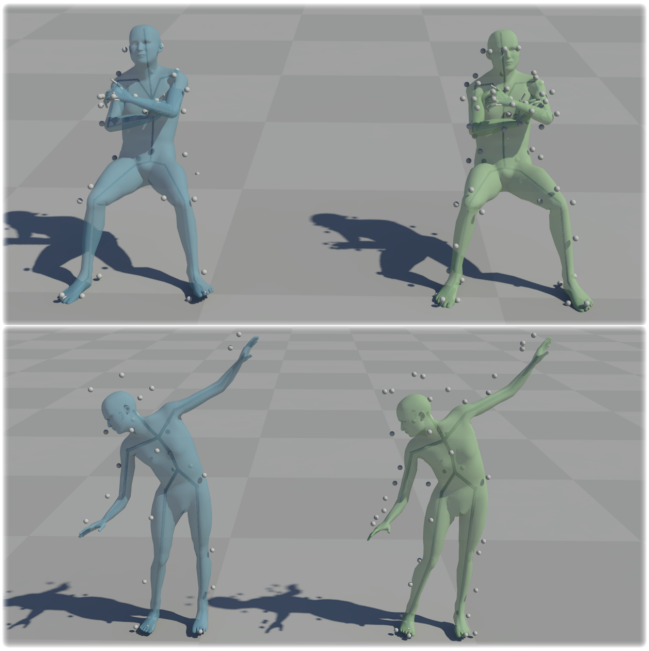

Fig. 9. Results of our method applied to motion where half of the markers have been removed. Left: Our Method. Right: Ground Truth. Top: Production Character. Bottom: Research Character.

6 RESULTS

In this section we show the results of our method on two different kinds of characters (see Table 1). The **Production** character is from a production game development environment and uses a proprietary motion capture database consisting of high quality motions captured for use in games, with a complex character skeleton including details such as finger joints. The **Research** character uses a more simple custom marker layout, with the skeleton structure and animation data from the BVH conversion of the CMU motion capture database [CMU 2013a]. Both characters are trained using roughly 5.3 million frames of animation at 120 fps which represents about 12 hours of motion capture data.

In Fig 1 and Fig 7 we show the results of our method using the Production character with several difficult-to-clean recordings, and compare our results to those produced by hand cleaning. Even when the data is very badly broken our method produces results difficult to distinguish from the hand cleaned result.

Next, we show some artificial experiments designed to demonstrate the robustness of our method. In Fig 8 we show our method using both the Production and Research characters applied when the data has been corrupted using our custom noise function. In Fig 9 we show our method on both characters using data where half of the markers have been removed, and in Fig 10 we show our results using data where all but one of the markers related to the right arm have been removed. In all cases we produce motion very close to the original ground truth.

Our method achieves high levels of precision - showing that it can be used both in production environments, when there is access

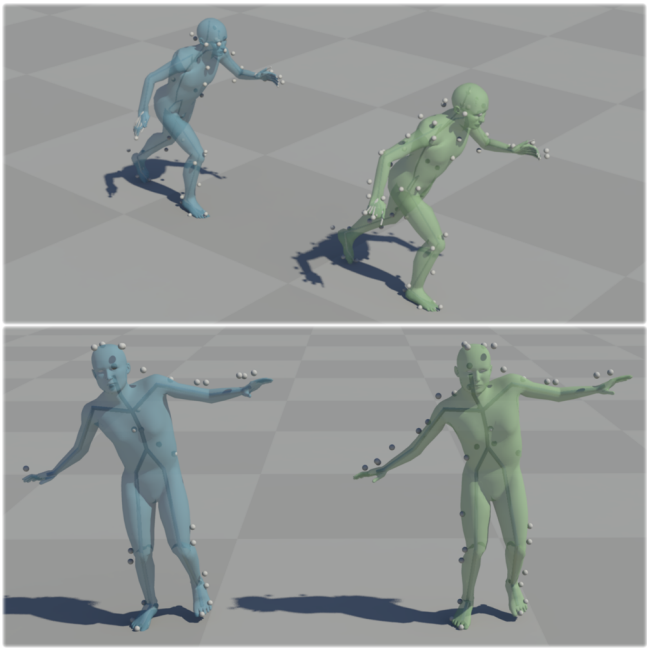

Fig. 10. Results of our method applied to motion where all but one of the markers related to the right arm have been removed. Left: Our Method. Right: Ground Truth. Top: Research Character. Bottom: Production Character.

to large quantities of high quality custom motion data, and in a research setting, using publicly available motion capture databases and a simple custom marker layout. For more results please see the supplementary video.

7 EVALUATION

In this section we evaluate the design decisions and performance of our approach. First, we compare our neural network structure against several alternative neural network structures including those that learn some kind of temporal prior such as CNNs and LSTM networks. Secondly, using the ResNet structure, we compare our corruption function against simpler noise functions such as Gaussian noise and Dropout. Finally we compare against commercial software, joint-space denoising methods, and non data-driven methods. For full details of the comparison please see Table 2. All comparisons are performed on the Production character using a set of test data (some of which is shown in Fig 1, Fig 7, and the supplementary video) using a Nvidia GeForce GTX 970. We train all networks until the training error converges or the validation error increases. To ensure a fair comparison, we use the same number of layers in all network structures and adjust the number of hidden units such that all neural network structures have roughly the same memory allowance. In Fig 13 we show the full distribution of errors of our method when applied to the test data set.

Our comparison finds that although many methods achieve a good level of accuracy, our simple ResNet structure achieves the best results on the test data. We found that complex neural network structures such as CNNs and LSTMs struggled to achieve the same level of accuracy and additionally can have slower inference and longer training time. For more discussion on why this might be the case please see Section 8. We find that commercial software has a low error on average but sometimes completely fails such as when there are mislabeled markers (see Fig 12), and that joint-space denoising methods also fail when the input contains too many errors (see Fig 11). Finally, we find that our noise function achieved the best results on the test set, indicating that the design of our custom noise function provides an important contribution to this task.

8 DISCUSSION

Temporal Prior. The only temporal prior encoded in our system is the Savitzky-Golay filtering descibed in Section 5 and used as a post-process to smooth the results. This basic prior is enough to achieve smoothness but in some situations we found it to be too strong and noticed over-smoothing in cases such as hard impacts. For this reason it makes sense to instead try to *learn* some form of temporal prior from the training data using a neural network structure such as a CNN or LSTM. Unfortunately, we found these more complex neural networks harder to train, and they were unable to achieve the desired accuracy even with extensive parameter tweaking. Additionally, we did not find the temporal prior learned by these methods to be significantly stronger than the Savitzky-Golay filtering, and still noticed fast, unnatural movements when markers quickly appeared and disappeared, as well as over-smoothed movements on hard impacts. For a visual comparison please see Fig 14.

After experimentation, we opted for the simplicity of a per-pose framework using a feed-forward neural network and filtering as a post-process. Firstly, this allowed us to train the neural network much faster, allowing for a deeper and wider structure. Secondly, using this type of network allowed us to use a simpler noise function which does not need to emulate the temporal aspect of motion capture data errors. Finally, this structured allowed for greater performance and simplicity in implementation at runtime as every pose can be processed individually and in parallel.

On the other hand, we still feel strongly that the temporal aspect of motion data should be useful for the task of fixing unclean motion data and as such, it seems that finding a neural network structure which encodes the temporal aspect of the data while still achieving the desired accuracy would be an excellent goal for future research.

Limitations. Like all data-driven methods, our method requires an excellent coverage in the data-space which can sometimes prove a challenge. As this data must be cleaned ahead of time, our method cannot be considered as automatic as existing non-data-driven methods. If some extreme poses are not covered in the training data, our method can potentially produce inaccurate results when it encounters these poses at runtime, even when there are no errors in the input data itself (see Fig 15). One way to counter this is to capture a very large range of movement recording where the actor tries to cover as many conceivable poses as possible in as short amount of time as possible. In this way, we predict the required training data can potentially be reduced from several hours to around 30-60 minutes.

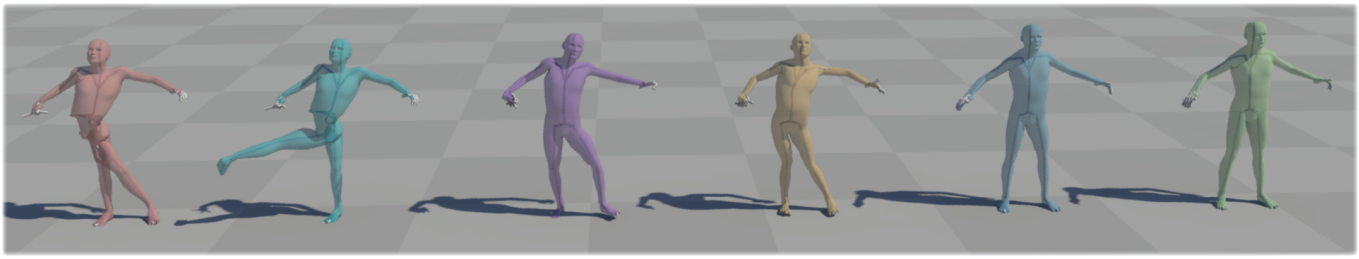

Fig. 11. Comparison with a selection of techniques which work in the space of the character joints. While these techniques work well in many cases they are often unable to reconstruct the original motion when the input joints contain too many bad errors. From left to right: Raw uncleaned data, Self-Similarity Analysis [Aristidou et al. 2018], Joint-Space LSTM Denoising [Mall et al. 2017]. Joint-Space CNN Denoising [Holden et al. 2015], Our method, Hand cleaned data.

Technique	Positional Error (mm)	Rotational Error (deg)	Inference (fps)	Memory (mb)	Training (hours)
ResNet (Our Method)	13.27	4.32	13000	75	18
FNN	15.28	4.98	13000	75	18
SNN	16.09	5.48	13000	75	18
ResCNN	25.06	7.66	3000	85	20
CNN	30.58	8.90	3000	85	20
SCNN	28.04	8.21	3000	85	20
GRU	19.22	7.66	350	75	30
LSTM	27.59	8.98	350	75	30
Commercial Software	12.94	4.33	-	-	-
Self-Similarity	51.89	4.75	15	-	-
Joint-Space CNN	19.70	5.57	3000	85	20
Joint-Space LSTM	48.96	35.06	350	75	30
Corrupt (Our Method)	13.27	4.32	-	-	-
Gaussian Noise	36.37	10.26	-	-	-
Uniform Noise	37.20	10.00	-	-	-
Dropout	108.64	14.84	-	-	-

Table 2. Comparison of different techniques applied to the test data. We measure the mean positional and rotational error for all joints across all frames in the test set, along with the inference time, training time, and memory usage. The techniques compared from top to bottom: Our Method (ResNet), Standard Feed-Forward Neural Network with ReLU activation function (FNN), Self-Normalizing Neural Network [Klambauer et al. 2017] (SNN), Residual Convolution Neural Network (ResCNN), Convolutional Neural Network (CNN), Self-Normalizing Convolutional Neural Network (SCNN), Encoder-Recurrent-Decoder GRU Network [Chung et al. 2014] (GRU), Encoder-Recurrent-Decoder LSTM Network [Fragkiadaki et al. 2015] (LSTM), Vicon Automatic Gap Filling [Vicon 2018] (Commercial Software). Self-Similarity Analysis [Aristidou et al. 2018] (Self-Similarity). Joint-Space Convolution Neural Network [Holden et al. 2015] (Joint-Space CNN), Joint-Space LSTM Network [Mall et al. 2017] (Joint-Space LSTM), Noise functions compared from top to bottom: Our Method (Corrupt), Simple Gaussian noise with a standard deviation of 0.1 applied to the input data (Gaussian Noise), Uniform noise with a range of 0.1 applied to the input data (Uniform Noise), Dropout [Srivastava et al. 2014] applied at each layer with a rate of 0.25 (Dropout).

Although our method often produces results which are visually hard to distinguish from hand cleaned data, it tends to have a residual error of a few millimeters when compared to the ground truth. In cases which require very precise data this could still be considered too large and can sometimes manifest in undesirable ways such as small amounts of visible foot sliding.

One common limitation to data-driven methods is that they are often extremely brittle to changes in the data set-up such as using different marker layouts or skeleton structures. A positive aspect of our algorithm is that it is relatively easy to change the marker layout: all it requires is a new set of marker configurations and for the network to be re-trained. As our algorithm drastically reduces the cost-per-marker of optical motion capture it would be interesting to see it used in conjunction with very large, dense marker sets covering the whole body such as in Park et al. [2006]. Regarding changes in the skeleton structure, our algorithm performs less well, and requires either retargeting all of the old data to the new skeleton or, at worst, collecting a whole new set of training data using the new skeleton.

Another limitation of our method is that our system's performance relies on the rigid body fitting process to be completed accurately, which in turn relies on the outlier removal process to perform well. If either of these stages fail, our method can sometimes produce worse than expected results (see Fig 16). One technique we discovered that helps this issue, is to add small perturbations to the transformations described by $\mathbf{F}$ during training. This ensures

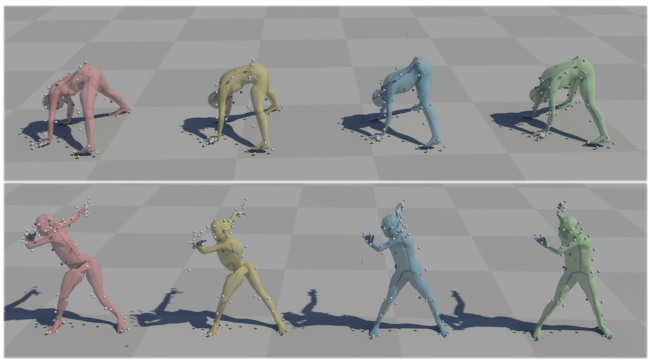

Fig. 12. Commercial software [Vicon 2018] has powerful tools for automatically filling gaps (Top) but tends to fail when encountering mislabeled markers (Bottom). From left to right: Raw uncleaned data, Automatic gap filling performed by Commercial Software, Our method, Hand cleaned data.

Fig. 13. The distribution of errors in the test data when using our approach. We find that roughly 90% of errors are less than 20 mm or 10 degrees in magnitude, and roughly 99.9% of errors are less than 60 mm or 40 degrees in magnitude.

the neural network generalizes well, even in situations where **F** is calculated inaccurately.

9 CONCLUSION

In conclusion, we present a data-driven technique aimed at replacing the solving part of the optical motion capture pipeline using a denoising neural network. Unlike existing techniques, our method is extremely robust to a large number of errors in the input data which removes the need for manual cleaning of motion capture data, reducing the burden of hand processing optical motion capture and greatly improving the potential throughput that can be achieved by an optical motion capture studio.

REFERENCES

Ijaz Akhter, Tomas Simon, Sohaib Khan, Iain Matthews, and Yaser Sheikh. 2012. Bilinear spatiotemporal basis models. *ACM Transactions on Graphics* 31, 2 (April 2012), 17:1–17:12. https://doi.org/10.1145/2159516.2159523

Andreas Aristidou, Daniel Cohen-Or, Jessica K. Hodgins, and Ariel Shamir. 2018. Self-similarity Analysis for Motion Capture Cleaning. *Comput. Graph. Forum* 37, 2 (2018).

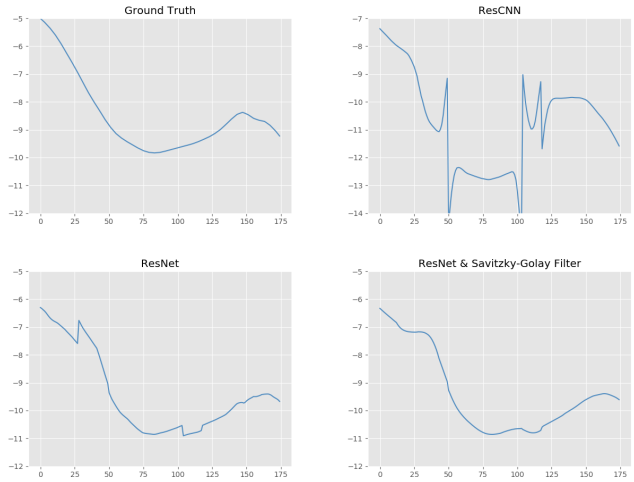

Fig. 14. Plots showing the x-axis position of the left knee joint when markers are quickly appearing and disappearing. Top-Left: Ground Truth. Bottom-Left: The raw output of our proposed ResNet architecture. As the ResNet calculates the output on a per-pose basis the trajectory contains small jumps when the markers quickly move in and out of visibility. Top-Right: The raw output of a convolutional ResNet architecture. Even though the convolutional neural network takes as input a window of the marker locations it appears unable to account for the behavior of disappearing markers and seems to interpret the disappearing markers as having large velocities towards or away from the origin, causing spikes in the resulting joint trajectories. Bottom-Right: ResNet output filtered using a Savitzky-Golay filter. The sharp movements are removed while the overall shape is mostly well-preserved.

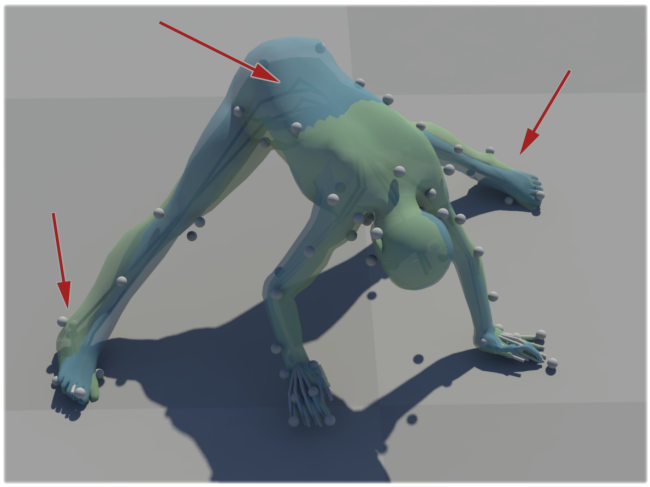

Fig. 15. In some extreme poses not covered in the training data, our network can produce less accurate results, even when there exists no problems with the input data. Blue: Our Method. Green: Ground Truth.

Andreas Aristidou and Joan Lasenby. 2013. Real-time marker prediction and CoR estimation in optical motion capture. *The Visual Computer* 29, 1 (01 Jan 2013), 7–26. https://doi.org/10.1007/s00371-011-0671-y

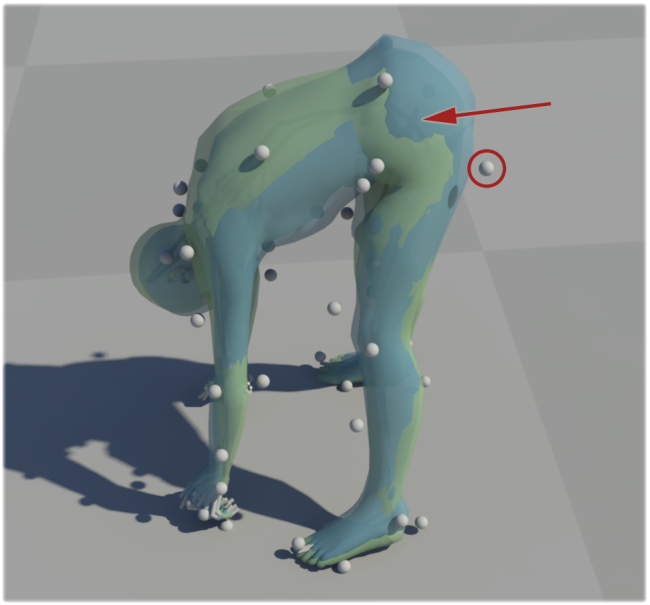

Fig. 16. In this example the badly positioned front-waist marker (circled in red) was not removed by the outlier removal process. This caused the rigid body fitting to produce an inaccurate local reference frame which resulted in the neural network positioning the hip joint too far back. Blue: Our Method. Green: Ground Truth.

Autodesk. 2016. MotionBuilder HumanIK. https://knowledge.autodesk.com/support/motionbuilder/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/MotionBuilder/files/GUID-44608A05-8D2F-4C2F-BDA6-E3945F36C872-htm.html. (2016).

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4, Article 97 (July 2017), 14 pages. https://doi.org/10.1145/3072959.3073708

Jan Baumann, Björn Krüger, Arno Zinke, and Andreas Weber. 2011. Data-Driven Completion of Motion Capture Data. In *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2011)*, Jan Bender, Kenny Erleben, and Eric Galin (Eds.). The Eurographics Association. https://doi.org/10.2312/PE/vriphys/vriphys11/111-118

Paul J. Besl and Neil D. McKay. 1992. A Method for Registration of 3-D Shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* 14, 2 (Feb. 1992), 239–256. https://doi.org/10.1109/34.121791

M. Burke and J. Lasenby. 2016. Estimating missing marker positions using low dimensional Kalman smoothing. *Journal of Biomechanics* 49, 9 (2016), 1854 – 1858. https://doi.org/10.1016/j.jbiomech.2016.04.016

Samuel Buss. 2004. Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. 17 (05 2004).

Jinxiang Chai and Jessica K. Hodgins. 2005. Performance Animation from Low-dimensional Control Signals. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 686–696. https://doi.org/10.1145/1186822.1073248

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (July 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* abs/1412.3555 (2014). arXiv:1412.3555 http://arxiv.org/abs/1412.3555

CMU. 2013a. BVH Conversion of Carnegie-Mellon Mocap Database. https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/cmu-bvh-conversion/. (2013).

CMU. 2013b. Carnegie-Mellon Mocap Database. http://mocap.cs.cmu.edu/. (2013).

Klaus Dorfmüller-Ulhaas. 2005. Robust Optical User Motion Tracking Using a Kalman Filter.

Yinfu Feng, Ming-Ming Ji, Jun Xiao, Xiaosong Yang, Jian J Zhang, Yueting Zhuang, and Xuelong Li. 2014a. Mining Spatial-Temporal Patterns and Structural Sparsity for Human Motion Data Denoising. 45 (12 2014).

Yinfu Feng, Jun Xiao, Yueting Zhuang, Xiaosong Yang, Jian J. Zhang, and Rong Song. 2014b. Exploiting temporal stability and low-rank structure for motion capture data refinement. *Information Sciences* 277, Supplement C (2014), 777 – 793. https://doi.org/10.1016/j.ins.2014.03.013

Tamar Flash and Neville Hogans. 1985. The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model. *Journal of neuroscience* 5 (1985), 1688–1703.

Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. 2015. Recurrent Network Models for Kinematic Tracking. *CoRR* abs/1508.00271 (2015). arXiv:1508.00271 http://arxiv.org/abs/1508.00271

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4, Article 138 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925975

Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning Motion Manifolds with Convolutional Autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs (SA '15)*. ACM, New York, NY, USA, Article 18, 4 pages. https://doi.org/10.1145/2820903.2820918

Hengyuan Hu, Lisheng Gao, and Quanbin Ma. 2016. Deep Restricted Boltzmann Networks. *CoRR* abs/1611.07917 (2016). arXiv:1611.07917 http://arxiv.org/abs/1611.07917

Alexander G. Ororbia II, C. Lee Giles, and David Reitter. 2015. Online Semi-Supervised Learning with Deep Hybrid Boltzmann Machines and Denoising Autoencoders. *CoRR* abs/1511.06964 (2015). arXiv:1511.06964 http://arxiv.org/abs/1511.06964

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.* 27, 4, Article 105 (Nov. 2008), 23 pages. https://doi.org/10.1145/1409625.1409627

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-Normalizing Neural Networks. *CoRR* abs/1706.02515 (2017). arXiv:1706.02515 http://arxiv.org/abs/1706.02515

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. 1998. Efficient BackProp. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. Springer-Verlag, London, UK, UK, 9–50. http://dl.acm.org/citation.cfm?id=645754.668382

Lei Li, James Mccann, Nancy Pollard, Christos Faloutsos, Lei Li, James Mccann, Nancy Pollard, and Christos Faloutsos. 2010. Bolero: A principled technique for including bone length constraints in motion capture occlusion filling. In *In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

Guodong Liu and Leonard McMillan. 2006. Estimation of Missing Markers in Human Motion Capture. *Vis. Comput.* 22, 9 (Sept. 2006), 721–728. https://doi.org/10.1007/s00371-006-0080-9

Xin Liu, Yiu-ming Cheung, Shu-Juan Peng, Zhen Cui, Bineng Zhong, and Ji-Xiang Du. 2014. Automatic motion capture data denoising via filtered subspace clustering and low rank matrix approximation. 105 (12 2014), 350–362.

Utkarsh Mall, G. Roshan Lal, Siddhartha Chaudhuri, and Parag Chaudhuri. 2017. A Deep Recurrent Framework for Cleaning Motion Capture Data. *CoRR* abs/1712.03380 (2017). arXiv:1712.03380 http://arxiv.org/abs/1712.03380

Sang Il Park and Jessica K. Hodgins. 2006. Capturing and Animating Skin Deformation in Human Motion. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. ACM, New York, NY, USA, 881–889. https://doi.org/10.1145/1179352.1141970

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryQu7f-RZ

Maurice Ringer and Joan Lasenby. 2002. *Multiple Hypothesis Tracking for Automatic Optical Motion Capture*. Springer Berlin Heidelberg, Berlin, Heidelberg, 524–536. https://doi.org/10.1007/3-540-47969-4_35

Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Deep Boltzmann Machines. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, David van Dyk and Max Welling (Eds.), Vol. 5. PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 448–455. http://proceedings.mlr.press/v5/salakhutdinov09a.html

Abraham. Savitzky and Marcel J. E. Golay. 1964. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry* 36, 8 (1964), 1627–1639. https://doi.org/10.1021/ac60214a047 arXiv:http://dx.doi.org/10.1021/ac60214a047

Alexander Sorkine-Hornung, Sandip Sar-Dessai, and Leif Kobbelt. 2005. Self-calibrating optical motion tracking for articulated bodies. *IEEE Proceedings. VR 2005. Virtual Reality, 2005.* (2005), 75–82.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from

Overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. 2011. Motion Reconstruction Using Sparse Accelerometer Data. *ACM Trans. Graph.* 30, 3, Article 18 (May 2011), 12 pages. https://doi.org/10.1145/1966394.1966397

Graham W. Taylor, Geoffrey E Hinton, and Sam T. Roweis. 2007. Modeling Human Motion Using Binary Latent Variables. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman (Eds.). MIT Press, 1345–1352. http://papers.nips.cc/paper/3078-modeling-human-motion-using-binary-latent-variables.pdf

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688

Alex Varghese, Kiran Vaidhya, Subramaniam Thirunavukkarasu, Chandrasekharan Kesavdas, and Ganapathy Krishnamurthi. 2016. Semi-supervised Learning using Denoising Autoencoders for Brain Lesion Detection and Segmentation. *CoRR* abs/1611.08664 (2016). arXiv:1611.08664 http://arxiv.org/abs/1611.08664

Vicon. 2018. Vicon Software. https://www.vicon.com/products/software/. (2018).

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.* 11 (Dec. 2010), 3371–3408. http://dl.acm.org/citation.cfm?id=1756006.1953039

Jun Xiao, Yinfu Feng, Mingming Ji, Xiaosong Yang, Jian J. Zhang, and Yueting Zhuang. 2015. Sparse Motion Bases Selection for Human Motion Denoising. *Signal Process.* 110, C (May 2015), 108–122. https://doi.org/10.1016/j.sigpro.2014.08.017

Zhidong Xiao, Hammadi Nait-Charif, and Jian J. Zhang. 2008. *Automatic Estimation of Skeletal Motion from Optical Motion Capture Data.* Springer Berlin Heidelberg, Berlin, Heidelberg, 144–153. https://doi.org/10.1007/978-3-540-89220-5_15

Junyuan Xie, Linli Xu, and Enhong Chen. 2012. Image Denoising and Inpainting with Deep Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 341–349. http://papers.nips.cc/paper/4686-image-denoising-and-inpainting-with-deep-neural-networks.pdf

Raymond A. Yeh, Chen Chen, Teck-Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson, and Minh N. Do. 2017. Semantic Image Inpainting with Deep Generative Models. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), 6882–6890.

Victor Brian Zordan and Nicholas C. Van Der Horst. 2003. Mapping Optical Motion Capture Data to Skeletal Motion Using a Physical Model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 245–250. http://dl.acm.org/citation.cfm?id=846276.846311

Received January 2018; revised January 2018; final version January 2018; accepted January 2018