Filter Guided Manifold Optimization in the Autoencoder Latent Space

Nate Lannan and Guoliang Fan
School of Electrical and Computer Engineering
Oklahoma State University, Stillwater, OK 74078
(nate.lannan,guoliang.fan)@okstate.edu

Abstract

An autoencoder is a class of neural network that is trained to output an accurate reproduction of the input while learning key lower dimensional features, otherwise known as a manifold. A lower dimensional representation of the original input, referred to as the latent space, encodes the intrinsic data structure over the manifold. This paper proposes filter-guided manifold optimization in the latent space of a convolutional autoencoder to recover noisy motion data collected by a depth sensor. Autoencoder output is smoothed using four traditional filters and employed as target motion data in an objective function. The difference between the actual output and target is minimized through stochastic gradient descent over the latent space, using manifold optimization to produce the expected smooth output. The advantage of this filter-guided approach over traditional filtering is that the resultant motion data still adheres to the manifold in the latent space learned by the autoencorder from training on motion data.

1. Introduction

The study of human motion is typically done with marker-based motion capture systems (Mocap) or markerless ones. Although the former are still considered the gold standard in most biomechanical studies and computer graphic applications, the latter, of which the most widely available are RGB-D sensor based, have several advantages. They are markerless and therefore less confining to the subject; they are not limited to a predefined area if they can be mobilized; and they are cost-effective and easy to use. Unfortunately these depth sensor-based systems are not as precise and detailed as the marker-based Mocap systems and suffer from erroneous skeleton estimation due to factors such as occlusion and lack of robustness. If these systems could be used to generate skeleton estimation and motion data that are reasonably or acceptably accurate, the benefits would be notable and valuable. Although sensor hardware will continue to improve, software approaches are more desirable and practical.

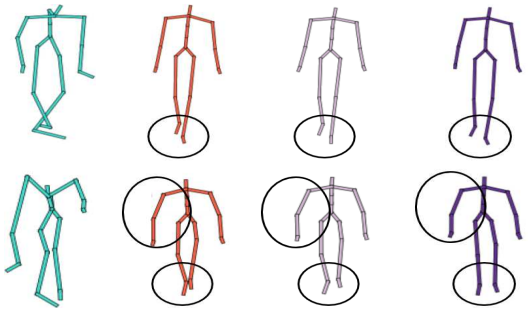

Figure 1. Progression of skeleton recovery: (a) Given noisy skeleton inputs, (b) Recovered skeleton without guided filtering, (c) Recovered skeleton with guided filtering, and (d) Ground truth skeleton.

Our research is aimed at improving skeleton data captured from an RGB-D sensor through deep learning. Many methods in current research capitalize on software development kits (SDKs) that have already been developed for various RGB-D sensors to estimate a 3D human skeleton. There are two approaches to improve skeleton data created by SDKs, i.e., filter-based, which attempt to smooth unnatural movements in joints [2, 9, 15], and learning-based, which use neural networks or pose graphs to characterize valid human motion and produce a cleaned version of the skeletal data [13, 6, 5, 12].

Inspired by [6, 5, 4], we propose a new filter-guided approach to skeleton data recovery that incorporates traditional filtering into autoencoder latent space optimization. The proposed approach capitalizes on the hidden layer latent space of the autoencoder and guides this space using a filtered version of the skeletal data, yielding a smoothed skeleton motion sequence, as shown in Figure 1, where two noisy inputs (a) are improved by autoencoder-based motion recovery [6] (b) and our method (c), creating results more comparable with the ground-truth data (d) captured by a marker-based Mocap system. The crux of this approach lies in that filter-guided latent space optimization balances the expected smoothness of output data with the learned lower dimensional manifold of human motion.

2. Related Work

We briefly review two kinds of approaches in related research, i.e., *filtering-based* and *learning-based*.

2.1. Filtering Methods

Most filtering methods have settled on some form of the Kalman filter to produce the best results, the Wiener Process Acceleration (WPA) Kalman filter in [2], the Tobit Kalman filter (TKF) in [9] and the extended Kalman filter in [15]. All of these filters were found to reduce the unnatural movements generated by self-occlusion. In addition, it was shown that the EKF is effective in smoothing out jitter effects and predicting the trajectory of a joint even after it is out of field-of-view, and the WPA Kalman reduces the latency associated with the Kalman Filtering process. Although these filtering methods did show some promise in improving the quality of skeleton estimation, they are still limited in dealing with large unnatural movements that are often caused by occlusion or erroneous estimation. If the filtering is accentuated to handle these cases, more of the higher frequency information of the subject's movement could be lost with more latency induced.

2.2. Learning Methods

The popular method for motion recovery is based on a large set of clean human motion data. A database generated from a Mocap system was used in [13] as a prior to correct erroneous movements in Kinect data, and then the corrected pose was dynamically filtered using a subject-specific model to match bone lengths and physical characteristics. This method heavily relies on *a priori* poses and the subject-specific model. On the other hand, neural networks prove an effective approach to improve the quality of noisy skeleton data. Two recurrent neural networks were used jointly in [12] to improve skeleton data where one neural net is trained on joint positions, and one on joint velocities. This method is supervised and involves a set of predefined poses for training. Autoencoder neural networks are commonly used in noise reduction and are a natural fit for the use in recovering noisy motion data from SDK estimation. A convolutional autoencoder neural network was used to learn a general motion manifold along low-dimensional latent space by training on a database of 10 hours of human motion data in [5, 6]. The autoencoder can then be used to correct real-world Kinect data by projecting the input into the latent space for motion denoising. This method does a good job correcting self-occlusion and data dropout but does not address some of the minor jitter effects of SDK estimation. Since traditional filtering methods are suitable to address this issue, one might consider a fusion of this learning method with traditional filtering.

3. Proposed Method

Our approach is based on a portion of the framework devised by Holden *et al*. [5]. We apply new filter based guidance to the latent (H) space of the autoencoder to improve motion recreation. This guidance acts on the compressed latent space as an objective function that can be optimized to generate smoother motion and reduced error when compared to ground truth.

3.1. Framework

The fundamental portion of this framework is a convolutional autoencoder used to correct skeletal motion data that has been corrupted in some manner (Figure 2). This autoencoder consists of an encoding step Φ and a decoding step Ψ. The encoder compresses or encodes the input data into a smaller subspace (H), extracting important characteristic features. The decoder then operates on the encoded data to reconstruct an approximation of the original data. This methodology has proven to be extremely useful in many denoising applications, as the output approximates a clean version of the input based on a learned manifold often unaffected by noise in the latent space.

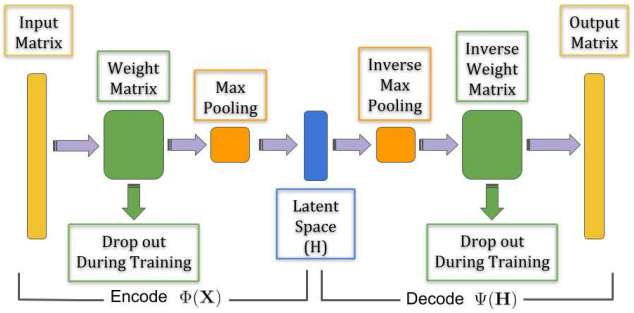

Figure 2. Network overview of the convolutional autoencoder.

The original framework by Holden *et al*. proposed a method to affect the style of recovered motion for use in synthesis of skeleton motion. This method involves creating a motion constraint that is applied to the latent space of the data. Using the latent space to modify the human motion has the advantage that the effects of the constraint do not violate the learned manifold, which is a representation of valid human motion. Objective functions are used to calculate an error in the visual space for all of the frames of motion. This error is then used to modify the latent space to converge toward the desired motion. This is done using gradient descent. The resultant modified latent space is then translated back to the visual space to produce a modified recovery. This process is presented in Figure 3.

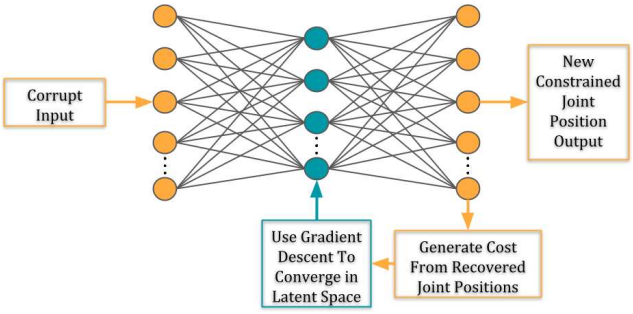

Figure 3. Using the latent space to constrain motion data.

3.2. Autoencoder

The input to this neural network has been modified from the original framework to only include the desired joint locations in the x, y, z coordinate space. The network trains on clips of 240 frames of motion data, which corresponds to 4 seconds worth of data, the amount of time needed to cover individual human motions [6]. Therefore, the input vector to the convolutional auto encoder is a matrix $\mathbf{X} \in \mathbb{R}^{240\times66}$.

Rather than have a fully connected neural network, the autoencoder is convolutional and the weight matrix is convolved along the frames of motion to encode dependencies of skeletal movement through time with a stride of 1 and no padding. The weight matrix, $\mathbf{W} \in \mathbb{R}^{256\times66\times25}$, produces convolutional filters corresponding to 25 frames of data. The result is then added to a bias vector $\mathbf{b} \in \mathbb{R}^{256}$, and a max pooling operation($S(\mathbf{x})$) is done to take the maximum value of each neighboring hidden units. This reduces the latent space by 1/2 in an attempt to down sample and concentrate on the most important features of the skeletal motion. The activation function used in the encode step is the rectified linear unit ($ReLU(\mathbf{x}) = max(\mathbf{x}, 0)$). The complete encoding step (1),

$$\Phi(\mathbf{X}) = ReLU(S(\mathbf{X} * \mathbf{W} + \mathbf{b})), \tag{1}$$

encapsulates the process of compressing the input to the latent space of the network. The output of this encode step is the latent space $\Phi(\mathbf{X}) = \mathbf{H}$, and is input to the decode step of the network.

The decode step is much the same as the encode step except for the fact that the weight matrix ($\widetilde{\mathbf{W}}$) is reflected across the temporal axis and transposed across the other two axes. The trained weights remain the same as the weights in the encode step, but rearranged to perform a deconvolution of the latent space. The decoder must also incorporate a method to undo the max pooling operation $S^{-1}(\mathbf{H})$, since max pooling is a lossy operation and the original data that was discarded cannot be recovered. In the training phase of the neural network framework, this inverse function is approximated by randomly setting one of the original units to the max value and the other to 0. However, in using the network to recover corrupted skeletal motion data, the max value is spread evenly to both units to reduce noise in the network. The result of this decode step (2),

$$\Psi(\mathbf{H}) = (S^{-1}(\mathbf{H}) - \mathbf{b}) * \widetilde{\mathbf{W}}, \tag{2}$$

is an approximation of the input $\mathbf{X}$ based on the lower dimensional representation contained in the latent space $\mathbf{H}$.

The network weights are initialized by the method described by Glorot and Bengio using the number of inputs to the network, *fan-in*, and the number of outputs of the network, *fan-out* [3]. This method sets minimum and maximum values for a uniform distribution within $[-r, r]$ (3),

$$r = \sqrt{\frac{6}{\textit{fan-in} + \textit{fan-out}}} \tag{3}$$

with which random numbers will be chosen to initialize the weights. The training of the network is discussed in further detail in Section 4.1.

3.3. Filter Guidance Latent Space Optimization

The latent space of the autoencoder has been trained to recreate human motion; therefore, the benefits of applying a filtering mechanism in this space capitalizes on traditional filtering methods as well as on the learned manifold, to recover corrupted human motion. Four filters are used to guide recovered motion through the autoencoder latent space, and three of these filters serve to consistently reduce noise in the skeleton and minimize the distance from ground truth data. The set of filters used to constrain motion in the latent space are an arithmetic average filter, a Savitzky-Golay filter, a Gaussian filter, and a Kalman filter. The filter guidance process involves calculating a theoretical output of the desired filter from the recovered data, and then reducing the difference between this theoretical output and the original recovered data.

The original recovered data is defined as a set of locations in the x, y, z coordinate space for each joint j, or $\mathbf{p}_j$. These joint positions are recovered for each frame, i, from the latent space, $\mathbf{H}_i$, using the autoencoder decode section as shown in Figure 3. Thus, $\mathbf{p}_j^{\mathbf{H}_i}$ is defined as joint position j recovered from frame i. This recovered motion data is then used to create a new matrix of joint positions, a target skeleton, for all frames by applying one of the four filtering techniques. Three of the target skeletons are constructed similarly, but each using a different kernel to define a set of C_k coefficient terms as shown in Figure 4. The filter kernel is applied as a weighted sum on joint j for a window of frames with size $[i - w, i + w]$ centered at frame i. This effectively builds a target skeleton over all frames based on the desired filter, where the value at frame i is a weighted

sum of adjacent frames $\sum_{k=-w}^{w} C_k \mathbf{p}_j^{\mathbf{H}_{i+k}}$. The cost function (4),

$$Cost(\mathbf{H}) = \sum_j \sum_i \left\| \mathbf{p}_j^{\mathbf{H}_i} - \sum_{k=-w}^{w} C_k \mathbf{p}_j^{\mathbf{H}_{i+k}} \right\|, \quad (4)$$

uses the Euclidean distance between the recovered joint matrix and the target joint matrix for optimization in the latent space. Because these matrices are implemented as tensors, we can then apply stochastic gradient descent in the latent space to minimize the objective function $Cost(\mathbf{H})$ as shown in Figure 3. After the minimization of the objective function, the latent space is fed through the decoder once again to generate new filter guided motion data.

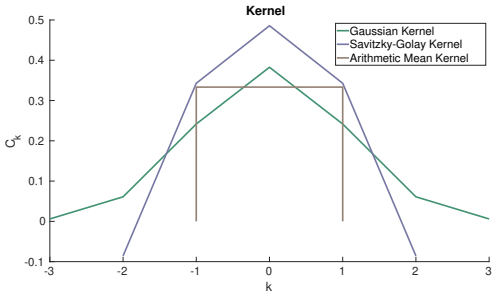

Figure 4. Filter kernels used in latent space optimization.

The arithmetic average kernel uses a window size of 3 ($w = 1$), and all of the coefficients are equal ($C_k = 1/3$). The Savitzky-Golay kernel models a filter with a window size of 5 and an order of 2, which produces the best results modeling a Savitzky-Golay filter [14]. The Gaussian kernel models a filter with a window size of 7 and approximates Gaussian values for this window size [8]. The window sizes for all three filters were arrived at experimentally based on performance. The fourth target skeleton, $\mathbf{q}_j^{\mathbf{H}_i}$, is calculated using a simplified Kalman filter process defined by Algorithm 1 [1]. In this algorithm, the target skeleton is calculated for each frame using the recovered original joints, $\mathbf{p}_j^{\mathbf{H}_i}$, as before. The initialization of process variance ($procVar$) was tuned experimentally, and the variance of the recorded values ($recVar$) was taken from the variance of a near-stationary joint. The target skeleton over all frames, $\mathbf{q}_j^{\mathbf{H}_i}$, is then used to calculate a Euclidean difference from the original recovered data, $\mathbf{p}_j^{\mathbf{H}_i}$, and summed over all frames and joints to form the objective function (5),

$$Kalman(\mathbf{H}) = \sum_j \sum_i \left\| \mathbf{p}_j^{\mathbf{H}_i} - \mathbf{q}_j^{\mathbf{H}_i} \right\|, \quad (5)$$

which serves to optimize the latent space of the network. This is done by minimizing $Kalman(\mathbf{H})$ using stochastic gradient descent over the latent space as shown in Figure 3. After minimization, the latent space is decoded to produce the filter guided motion data.

Algorithm 1: Creation of a target skeleton using a Kalman filter.

Result: Joint vector $\mathbf{q}_j^{\mathbf{H}_i}$ across all frames of motion

$currPos = \mathbf{p}_j^{\mathbf{H}_0}$;
$dt = 1/30$;
$dpos = (\mathbf{p}_j^{\mathbf{H}_1} - \mathbf{p}_j^{\mathbf{H}_0})/dt$;
$\mathbf{q}_j^{\mathbf{H}_0} = currPos$;
$currVar = 0$;
$procModel = dpos * dt$;
for *frame k=1:end* **do**
  $prior = currPos + procModel$;
  $priorVar = currVar + procVar$;
  $L = \mathbf{p}_j^{\mathbf{H}_k}$;
  $currPos = \frac{priorVar*L+recVar*prior}{priorVar+recVar}$;
  $currVar = \frac{priorVar*recVar}{priorVar+recVar}$;
  $\mathbf{q}_j^{\mathbf{H}_k} = currPos$;
  $dpos = (\mathbf{p}_j^{\mathbf{H}_{k+1}} - \mathbf{p}_j^{\mathbf{H}_k}) * dt$;
  $procModel = dpos * dt$;
end

4. Experimental Results

4.1. Data and Training

Experimentation was carried out using a neural network implemented in Theano [16] and trained using over 10 hours of motion capture skeleton data [5]. The dataset was collected from four sources, the CMU Graphics Lab Motion Capture Database [17], the Berkeley Multimodal Human Action Database (MHAD) [11], the Mocap Database HDM05 [10], and the data originally collected in [5, 4]. The data from these sources was retargeted to a skeleton of common structure based on the CMU database to ensure skeleton consistence. The skeletal structure retargeting was done using inverse kinematics on matching joint angles [18].

After this retargeting was completed, the skeleton joint angles were converted into joint positions relative to joint 0 (a projection of the central hip joint on to the floor), and the skeleton was scaled by subtracting the mean from the data and dividing by the standard deviation of each joint position. The joint positions were then used as an input matrix to the autoencoder, and the autoencoder was trained to recreate the input. This was done by reducing the network loss function through stochastic gradient descent and incrementally updating the weights and biases using the Adaptive Moment Estimation (Adam) algorithm [7] with a learning rate of 0.001 and two moments of 0.9 and 0.999. The network utilized dropout regularization to deter overfitting with a dropout rate of 0.2. The network loss function was arrived at by squared error between the input matrix $\mathbf{X}$ and the output of the autoencoder (6),

$$Loss(\mathbf{X}, \theta) = \|\mathbf{X} - \Psi(\Phi(\mathbf{X}))\|_2^2 + \alpha \|\theta\|_1, \quad (6)$$

where the second term is used as sparsity regularizer on the weights and biases of the network(θ), and is controllable by adjusting α.

4.2. Test Data

The network was used to recover two types of data. The first was purely simulated data in that the original motion capture skeleton data used to train the network was corrupted by randomly setting the values of joint positions to a zero position at a rate of 50%. This acts as drop-out for half the data of the original motion capture data and mimics the jerky transient motion of data corrupted by momentary self-occlusion. The disadvantage of this type of data is that it is derivative of the original training data, even though 50% of it has been removed. The second type of data is real-world data captured by Holden *et al*. [4], in which data collected from a Kinect is time synced to data from a motion capture system to act as ground truth. The Kinect data has been retargeted and normalized for use with the network, and a real world comparison can be made between the Kinect recovered skeleton and the time synced motion capture data.

4.3. Filter Guidance Evaluation

Each of the filter guidance processes described in section 3 were evaluated by calculating the Euclidean distance of each joint position from the ground truth joint position for each frame of motion. A distance from ground truth was then taken over all frames of motion for each joint. First, the filters were assessed over 240,000 frames of simulated corrupted motion data and compared in Table 1. The results of the Kalman filter have been omitted as the Kalman filter performed poorly in comparison with the other three filters. This is likely due to the non-linear nature of the data. The implementation of an extended Kalman filter might produce more favorable results. We see that each of the three filters improve upon the recovered data in which guidance has not been applied, with the slight exception of some of the inner joints (2,10,11), consisting of the hips, and spine. This is because these joints were rarely affected by the simulated noise since they did not move away from their zero positions as much as the external joints, and when they were set back to their zero positions randomly, the magnitude of their movement was minimal.

More importantly, the filters were assessed over 148,800 frames of real-world Kinect data synced with motion capture ground truth. The recovered unguided Kinect data is compared against the three filter guided recoveries. Here in Table 2 we see that all three filters improve upon the recovery of the autoencoder, yielding a result that is closer to ground truth. The improvement across joints is more uniform since the noise from the real-world data tends to affect all of the joints equally. The question now arises, does filter guidance in the latent space provide any benefit that a traditional filter of the same type does not? In Table 3 we compare traditional filters applied to the recovered skeleton versus the guidance process. We see that the filter guidance does indeed outperform the traditional filters, in all but one case due to the fact that they can influence the skeleton to approach a filtered signal while maintaining the natural human motion learned in the autoencoder manifold.

A qualitative comparison of four joint positions across

Joints	Corrupted Data	No Filter Guidance	Arithmetic Average	Savitzky-Golay	Gaussian
1	9.205	2.514	**2.365**	2.443	2.455
2	9.856	**2.853**	2.918	2.966	2.976
3	16.862	5.136	**4.758**	4.772	4.767
4	22.678	6.825	6.114	**6.113**	**6.113**
5	23.552	7.433	**6.354**	6.369	6.359
6	9.888	2.984	**2.908**	2.961	2.971
7	17.354	5.221	5.068	**5.061**	**5.061**
8	23.158	6.745	**6.037**	6.041	6.042
9	23.922	7.292	**6.402**	6.415	6.408
10	10.565	**2.885**	2.919	2.994	3.012
11	12.646	**3.337**	3.339	3.401	3.417
12	14.816	3.785	**3.747**	3.786	3.796
13	16.818	4.355	**4.337**	4.348	4.356
14	14.311	3.990	**3.854**	3.882	3.886
15	20.560	6.101	5.399	5.396	**5.393**
16	27.900	7.821	6.228	**6.217**	6.218
17	29.868	8.384	6.642	**6.619**	6.621
18	14.478	4.176	**4.030**	4.062	4.070
19	22.412	6.579	5.519	**5.511**	5.515
20	30.196	8.456	6.413	**6.391**	**6.391**
21	31.852	8.996	6.821	**6.795**	**6.792**
Ave.	19.186	5.518	**4.865**	4.883	4.887

Table 1. Mean distance from ground truth (cm) of joint positions over 240,000 frames of simulated data. Recovered data without filter guidance is compared with filter-guided results using three different filters, where the lowest value is highlighted in green.

Joints	Corrupted Data	No Filter Guidance	Arithmetic Average	Savitzky-Golay	Gaussian
1	7.372	7.110	**4.219**	4.313	4.319
2	6.905	6.350	**4.697**	4.738	4.743
3	8.617	8.750	**6.773**	6.816	6.814
4	8.459	8.751	**7.651**	7.682	7.680
5	9.137	9.060	**8.188**	8.221	8.219
6	6.727	6.309	**5.001**	5.035	5.037
7	6.919	6.806	6.378	6.340	**6.338**
8	8.115	7.709	6.888	6.896	**6.884**
9	8.949	8.314	**7.645**	7.653	**7.645**
10	7.675	7.352	**4.742**	4.828	4.833
11	8.365	8.339	**5.714**	5.797	5.801
12	9.364	8.895	**6.617**	6.677	6.679
13	10.397	10.022	**7.929**	7.974	7.973
14	10.332	9.776	**7.961**	7.987	7.988
15	9.577	9.396	**8.281**	8.290	8.284
16	10.477	10.219	8.854	8.846	**8.843**
17	10.893	10.854	9.666	9.641	**9.637**
18	10.940	10.088	**8.587**	8.591	8.595
19	10.309	10.241	**9.000**	9.015	9.026
20	10.907	11.123	**9.904**	9.913	9.924
21	11.268	11.570	10.673	**10.659**	10.670
Ave.	9.129	8.906	**7.398**	7.424	7.425

Table 2. Mean distance from ground truth values (cm) of joint positions over 148,800 frames from real-world Kinect data synced with motion capture ground truth. Recovered data without filter guidance is compared with filter-guided results using three different filters, where the lowest value is highlighted in green.

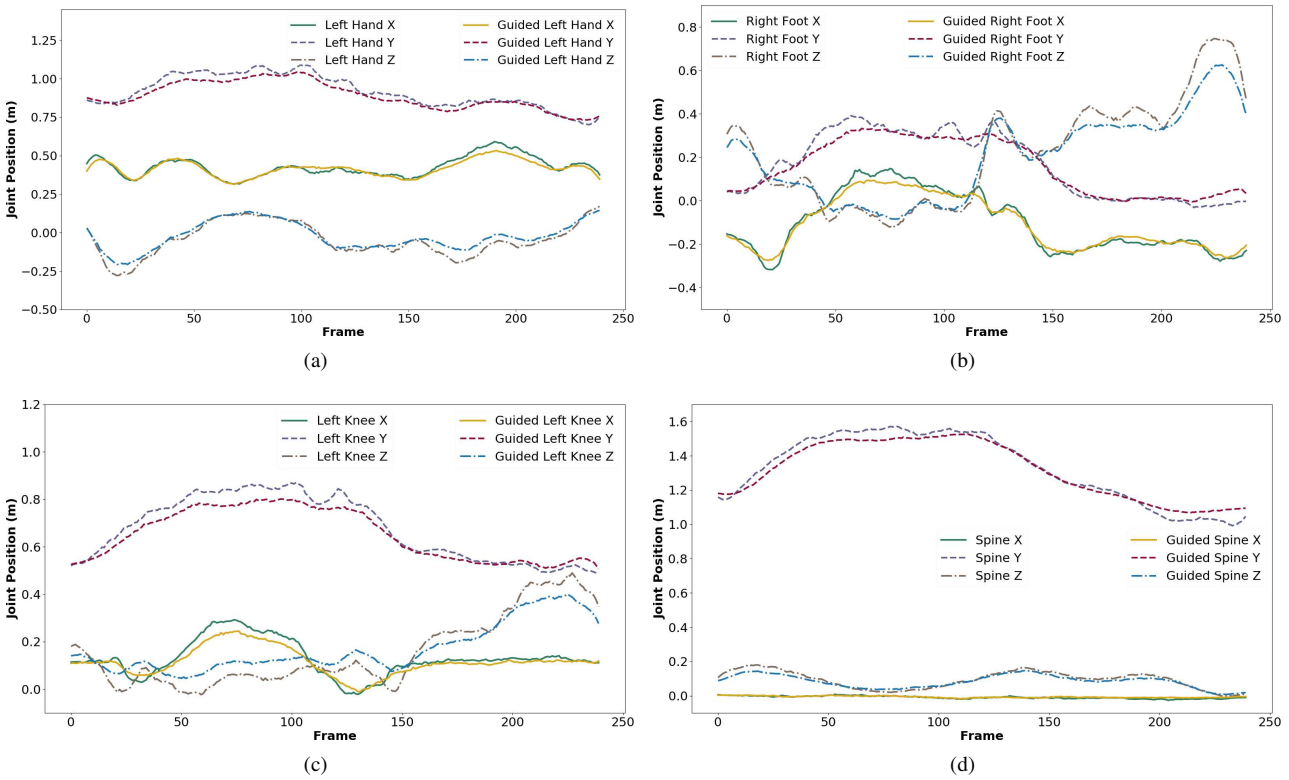

Figure 5. Smoothing effect on joint positions: x, y, z coordinates with and without guidance for (a) the left hand, (b) the right foot, (c) the left knee, and (d) the spine. The results are more evident in the extremities as the internal joints tend to stay steady.

Methods	Simulated Data	Real Data
Traditional arithmetic average	5.307	8.846
Arithmetic average filter guided	**4.865**	**7.398**
Traditional Savitzky-Golay filter	5.512	8.909
Savitzky-Golay filter guided	**4.883**	**7.424**
Traditional Gaussian filter	5.196	8.800
Gaussian filter guided	**4.887**	**7.425**
Traditional Kalman filter	**5.621**	8.948
Kalman filter guided	6.521	**8.664**

Table 3. Comparison between traditional filtering methods and the proposed filter-guided approach in terms of the mean distance from ground truth (cm) averaged over 21 joints.

240 frames of motion is given in Figure 5, in which the smoothing effect of the filter guidance process on the left hand, right foot, left knee, and central spine joints is apparent. Here the x,y, and z coordinates have been displayed before and after the guidance process. Clearly the filter guidance has reduced the noise in the unguided recovered signal and produced a smoother approximation to human motion. This result is more noticeable for the external joints as the nature of the induced noise causes them to move farther, and the internal joints such as the spine remain relatively steady, with minimal high frequency content.

5. Conclusion and Future Work

We have presented a new filter-guided latent space optimization method to improve the performance of the convolutional autoencoder that is used to recover corrupted human motion data or low quality skeleton data from an RGB-D sensor and SDK tools. An objective function based on the difference between an expected filtered output and the unfiltered recovered one was developed to optimize the latent space, smoothing the human motion data while forcing it to adhere to the learned manifold. The proposed method outperforms the original autoencoder and traditional filters, due to the fact that we are capitalizing on the nature of the latent space which is an encoded low dimensional representation of human motion. This filter-guided method is appealing for many real-world applications involving human motion data by improving RGB-D SDK outputs. Future work will focus on the study on using extended Kalman filters to guide latent space optimization for further improvement.

Acknowledgment

This work is supported in part by the US National Institutes of Health (NIH) Grant R15 AG061833 and the Oklahoma Center for the Advancement of Science and Technology (OCAST) Health Research Grant HR18-069.

References

[1] G. Bishop, G. Welch, et al. An introduction to the Kalman filter. In *Proceedings of SIGGRAPH, Course*, 8(27599-23175):41, 2001. 4

[2] M. Edwards and R. Green. Low-latency filtering of Kinect skeleton data for video game control. In *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand*, pages 190–195. ACM, 2014. 1, 2

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 3

[4] D. Holden. The orange duck, 2019. Data retrieved from theorangeduck.com, http://theorangeduck.com/. 1, 4, 5

[5] D. Holden, J. Saito, and T. Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics*, 35(4):138:1–138:11, July 2016. 1, 2, 4

[6] D. Holden, J. Saito, T. Komura, and T. Joyce. Learning motion manifolds with convolutional autoencoders. In *Proceedings of SIGGRAPH Asia 2015 Technical Briefs*, SA '15, pages 18:1–18:4, New York, NY, USA, 2015. ACM. 1, 2, 3

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[8] T. Lindeberg. Scale-space for discrete signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):234–254, 1990. 4

[9] K. Loumponias, N. Vretos, P. Daras, and G. Tsaklidis. Using Kalman filter and Tobit Kalman filter in order to improve the motion recorded by Kinect sensor II. In *Proceedings of the 29th Panhellenic Statistics Conference*, 2016. 1, 2

[10] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation mocap database HDM05. Technical Report CG-2007-2, Universität Bonn, June 2007. 4

[11] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy. Berkeley MHAD: A comprehensive multimodal human action database. In *Proceedings of 2013 IEEE Workshop on Applications of Computer Vision (WACV)*, pages 53–60. IEEE, 2013. 4

[12] Y. Park, S. Moon, and I. H. Suh. Tracking human-like natural motion using deep recurrent neural networks. *CoRR*, abs/1604.04528, 2016. 1, 2

[13] P. Plantard, H. P. H. Shum, and F. Multon. Filtered pose graph for efficient Kinect pose reconstruction. *Multimedia Tools and Applications*, 76(3):4291–4312, Feb 2017. 1, 2

[14] R. W. Schafer et al. What is a Savitzky-Golay filter. *IEEE Signal Processing Magazine*, 28(4):111–117, 2011. 4

[15] J. Shu, F. Hamano, and J. Angus. Application of extended Kalman filter for improving the accuracy and smoothness of Kinect skeleton-joint estimates. *Journal of Engineering Mathematics*, 88(1):161–175, Oct 2014. 1, 2

[16] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. 4

[17] C. M. University. Carnegie Mellon Mocap Database, 2019. Data retrieved from Carnegie Mellon Mocap Database, http://mocap.cs.cmu.edu/. 4

[18] K. Yamane and Y. Nakamura. Natural motion animation through constraining and deconstraining at will. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):352–360, July 2003. 4