Tucker decomposition with a temporal regularization for gap recovery in 3D motion capture data

Citation for the original published paper (version of record):
Mohaoui, S., Dmytryshyn, A. (2026). Tucker decomposition with a temporal regularization for gap recovery in 3D motion capture data. Applied Mathematics and Computation, 522.
http://dx.doi.org/10.1016/j.amc.2026.129996

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Full Length Article

Tucker decomposition with a temporal regularization for gap recovery in 3D motion capture data

Souad Mohaoui [a,*], Andrii Dmytryshyn [b,a]

[a] School of Science and Technology, Örebro University, Örebro, Sweden
[b] Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

ARTICLE INFO

Keywords:
Gap-filling
Missing markers
MoCap systems
Tucker decomposition
Tensor recovery
Proximal BCD algorithm

ABSTRACT

The gap-filling problem in motion capture (MoCap) data poses a significant challenge in marker-based MoCap systems. These gaps occur due to missing markers during motion recording. MoCap sequence, a time series data characterized by high dimensionality and temporal dependencies, requires accurate recovery of missing markers to ensure smooth motion representation. Tensor decomposition is an effective solution that leverages the multi-way structure of MoCap data. This paper proposes two gap-filling algorithms based on Tucker decomposition, namely Tucker and TuckerTNN. Given the high-dimensional nature of MoCap data, traditional smoothness regularization methods, such as gradient-based techniques, are computationally expensive. Therefore, we introduce the temporal nuclear norm in TuckerTNN as an alternative regularization technique, providing a more efficient solution for large-scale datasets. Both models are minimized using the proximal block coordinate descent (Prox-BCD) method. We evaluated the proposed algorithms using motion capture sequences from the publicly available HDM05 dataset. Our results show that Tucker and TuckerTNN consistently outperform existing approaches, such as CP and SparseCP, in accuracy and efficiency, with TuckerTNN offering the best trade-off between the two.

1. Introduction

The ability to monitor and analyze object motion, especially human movement, plays a crucial role across various fields. Motion Capture (MoCap) technologies serve as the primary means of recording three-dimensional movement over time. These systems rely on optical sensors or markerless computer vision approaches to acquire high-precision spatiotemporal data. With advancements in computational tracking and biomechanical modeling, MoCap has become essential in diverse fields, such as clinical rehabilitation (e.g., gait analysis and prosthetic development) [3,4], sports performance enhancement [5,6], and humanoid robotics for motion synthesis [7]. Although markerless techniques are advancing, marker-based MoCap systems remain prevalent in human motion studies. Such frameworks employ retroreflective markers placed on key body joints or segments, triangulated via synchronized infrared camera arrays. The collected positional data undergo computational processing to generate a digital skeletal representation, enabling precise reconstruction of joint kinematics and segmental orientations.

A major drawback of marker-based MoCap systems is the occurrence of missing markers during the recording process. Markers can become occluded, misidentified, or lost due to factors such as overlapping body parts, camera angles, or changes in lighting conditions. This results in incomplete data and gaps in the recorded motion. These gaps disrupt the continuity of the motion recordings, making

[*] Corresponding author.
E-mail addresses: souad.mohaoui@oru.se (S. Mohaoui), andrii@chalmers.se, andrii.dmytryshyn@oru.se (A. Dmytryshyn).

https://doi.org/10.1016/j.amc.2026.129996
Received 4 February 2025; Received in revised form 8 September 2025; Accepted 28 January 2026
Available online 6 February 2026

it difficult to reconstruct an accurate representation of human movement. As a result, gap reconstruction in MoCap data is essential for maintaining data integrity and enabling reliable motion analysis.

The gap-filling problem in motion capture data has been predominantly addressed using matrix-based frameworks. In this representation, a motion sequence is typically organized as a matrix where rows correspond to the number of frames and columns represent the spatial coordinates of all markers or joints. Matrix-based methods have been extensively developed and widely adopted for addressing the gap-filling problem [1,2,8–19]. These approaches leverage various matrix completion and factorization techniques, including low-rank matrix recovery, principal component analysis, and non-negative matrix factorization. However, this matrix representation inherently loses the natural multidimensional structure of MoCap data. Moreover, matrix-based methods may fail to fully capture the complex dependencies and structural relationships that exist across different modes of the data.

Tensor-based approaches for gap filling in MoCap data remain largely unexplored in the literature. Tensors, a generalization of matrices, provide a natural tool for coherently representing multimodal datasets that exhibit complex multidimensional structures. Unlike matrices, which are restricted to two dimensions, tensors can encapsulate data across three or more dimensions. To the best of our knowledge, only one recent work has introduced a tensor-based framework for gap-filling problems in MoCap data [20], where three CP decomposition algorithms are proposed. CP decomposition represents a tensor as a sum of rank-one component tensors, making it particularly suitable for interpreting multi-way data. However, CP decomposition has limitations in capturing interactions between different dimensions of the MoCap data. The present work represents the second tensor-based approach in this domain, introducing Tucker decomposition as an alternative that can better capture the multilinear relationships inherent in MoCap data. While Tensor-Train (TT) decomposition represents another tensor factorization approach, it is designed mainly for high-order tensors ($4 \leq N$), where it achieves significant computational advantages by addressing the curse of dimensionality. TT decomposition reduces both storage and computational costs through its sequential matrix representation, where the number of parameters scales linearly with tensor order rather than exponentially as in Tucker decomposition. However, for three-dimensional tensors like our MoCap data, TT decomposition offers no computational benefits over Tucker decomposition.

In addition, considering the inherent properties of motion data contributes to enhancing the reconstruction of missing markers. MoCap sequences consist of time series data, with a large temporal dimension. They exhibit temporal continuity, ensuring smooth transitions in position, velocity, and acceleration as human motion evolves. Explicitly modeling this temporal smoothness using techniques like total variation or graph-based regularization can be computationally expensive, especially when handling high-dimensional MoCap data.

This paper considers Tucker decomposition to handle the gap-filling problem in MoCap recordings. The Tucker model decomposes a tensor into a core tensor multiplied by matrices along each mode, thus capturing interactions across different dimensions. While MoCap data is large in the time dimension, the spatial dimensions-such as the number of markers and their positions-are relatively small. Tucker decomposition provides greater flexibility by allowing different ranks for each mode, enabling it to capture more complex, multi-way interactions in high-dimensional data. We adopt the Tucker decomposition framework and introduce two MoCap gap-filling algorithms. The first algorithm employs Tucker decomposition as a baseline method, while the second extends it by incorporating temporal smoothness through a nuclear norm regularization on the temporal mode. MoCap data possesses temporal continuity as the motion changes smoothly over time, see Fig. 7. Thus, the temporal nuclear norm is used as an alternative to traditional smoothness regularization. Rather than directly penalizing temporal differences, the nuclear norm encourages a low-rank representation of the temporal mode using a small number of dominant temporal patterns in the motion data. This approach inherently enforces smooth evolution over time, improving the accuracy and continuity of the reconstructed motion.

The structure of this paper is as follows: Following the introduction, Section 2 provides a review of related work. Section 3 outlines the key notations used throughout the paper. Section 4 details the proposed Tucker and TuckerTNN tensor completion methods, while Section 5 introduces the rank estimation technique for tensor decomposition. In Section 6, we present and analyze numerical simulations conducted on various MoCap sequences from the online Motion Capture Database HDM05[1]. Finally, Section 7 concludes the paper.

2. Related works

Different motion reconstruction methods have been designed to recover incomplete motion data. Interpolation methods, such as linear interpolation, spline interpolation, and Kalman filtering, are the standard techniques that estimate the missing markers based on the available marker positions and some assumptions or constraints [21–25]. These methods use the neighboring available markers and usually rely on the continuity of the motion sequence, where a manual intervention is often needed. Data-driven methods are another approach to solving the missing marker problem [26,27]. These methods typically employ a database of similar motions to recover the missing entries. These approaches may fail to predict the missing markers if the informative markers are unavailable. The skeleton-driven approaches employ kinematic or bone-length constraints [24,28,29]. However, skeleton-based methods heavily rely on the accurate position of markers on the human body. Slight variations in marker placement between different subjects or sessions can affect the performance and generalization of the approach. Principal component analysis-based approaches [17–19] essentially exploit the linear or nonlinear correlations of the motion matrix to estimate the missing gap in the MoCap data.

Similarly, low-rank matrix completion methods are also employed to recover missing markers. The success of low-rank frameworks stems from their ability to model the dependence of joint movements, which naturally restricts motion data to a lower-dimensional

[1] https://resources.mpi-inf.mpg.de/HDM05/index.{html}

structure. Therefore, by capitalizing on the inherent low-dimensional structure of human motion, these methods effectively impute gaps in motion sequences [8,9].

Biomechanical constraints are frequently integrated into these frameworks to enhance the reconstruction accuracy. For instance, enforcing rigid bone lengths during completion prevents inconsistencies such as incorrect bone lengths and unrealistic skeletal reconstruction [10,11]. Temporal smoothness, another critical constraint, addresses the continuous evolution of position, velocity, and acceleration in human motion recordings. By incorporating temporal consistency into low-rank models, abrupt trajectory discontinuities or unrealistic motion transitions are avoided [11–13]. Besides, to enforce skeletal integrity and motion continuity across frames approaches such as [14] considered spatial and temporal constraints simultaneously. For highly dynamic or nonlinearly correlated motions-such as those involving rapid pose changes or environmental variability-kernel-based nonlinear low-rank completion method is proposed [16]. This approach employs multiple kernel learning to model complex motion patterns, outperforming linear models in unpredictable or high-variability conditions. Nonnegative matrix factorization leverages the non-negative characteristics of MoCap data (e.g., joint angles or marker distances) to produce sparse reconstructions [15].

Tensors are useful tools for modeling complex interactions and preserving inherent relationships within high-order datasets for accurate pattern and dependency analysis. Analogous to matrix factorizations, tensor decompositions aim to break down complex data structures into simpler, more interpretable components, enabling precise extraction of significant features. Tensor decompositions are essential for overcoming the information loss that occurs when data is flattened. Several formats of tensor decomposition have been studied, such as CANDECOMP/PARAFAC (CP) [30], Tucker [31], and tensor train [32]. Since MoCap data has a complex, multidimensional structure, with dependencies across time, space, and joint positions, the gap-filling problem is an ideal application for low-rank tensor decomposition. To the best of our knowledge, the work in [20] represents the only prior tensor-based approach for MoCap gap recovery, introducing three distinct completion techniques utilizing CP decomposition. One method employs a conventional CP decomposition model, while the other two are designed to enhance reconstruction by enforcing smoothness and sparsity constraints, effectively adapting to the varying complexity of MoCap data in different contexts.

3. Basic notations

Tensors are denoted using Euler script (e.g., $\mathcal{X}$), matrices are represented by bold uppercase letters (e.g., $\mathbf{X}$), vectors by bold lowercase letters (e.g., $\mathbf{x}$), and scalars by standard lowercase letters (e.g., x). $\mathbf{x}_i$ refers to the i-th column of the matrix $\mathbf{X}$. The order of a tensor corresponds to the number of its dimensions. A first-order tensor represents a vector, while a second-order tensor corresponds to a matrix. More generally, an N-order tensor $\mathcal{X}$ with dimensions $I_1 \times I_2 \times \cdots \times I_N$ is an N-dimensional array in $\mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$. The indexing conventions are as follows: the element at position $(i_1, i_2, \dots, i_N)$ in a tensor $\mathcal{X}$ is written as $\mathcal{X}_{i_1 i_2 \dots i_N}$, the matrix entry at (i_1, i_2) is represented as $\mathbf{X}_{i_1 i_2}$. For a positive integer N, we use the notation $[N] := \{1, 2, \dots, N\}$. Following [33], we define the notions below.

Definition 1. Given two tensors $\mathcal{X}, \mathcal{Z} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$, define their inner product as follows

$$\langle \mathcal{X}, \mathcal{Z} \rangle = \sum_{i_1,\dots,i_N} \mathcal{X}_{i_1\dots i_N} \cdot \mathcal{Z}_{i_1\dots i_N}. \tag{1}$$

Definition 2. Define

$$\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle} \tag{2}$$

to be a Frobenius norm of a given tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$.

Definition 3. The Khatri-Rao product, denoted by $\odot$, of two matrices $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_M] \in \mathbb{R}^{K\times M}$ and $\boldsymbol{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_M] \in \mathbb{R}^{J\times M}$, is defined by

$$\boldsymbol{X} \odot \boldsymbol{Z} = \begin{bmatrix} \boldsymbol{x}_1 \otimes \boldsymbol{z}_1, & \boldsymbol{x}_2 \otimes \boldsymbol{z}_2, & \cdots & , \boldsymbol{x}_M \otimes \boldsymbol{z}_M \end{bmatrix} \in \mathbb{R}^{KJ\times M}, \tag{3}$$

and $\otimes$ refers to the Kronecker product.

Definition 4. Given a tensor $\mathcal{Z} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$. Define the mode-$n$ unfolding (matricization) of $\mathcal{Z}$ as the matrix $\mathbf{Z}_{(n)} \in \mathbb{R}^{I_n\times M}$, with $M = \prod_{j\neq n}^N I_j$, obtained by converting a tensor to a matrix according to the mode-n where $\mathcal{Z}_{i_1\dots i_N}$ corresponds to $\mathbf{Z}_{i_n,s}$, with

$$s = 1 + \sum_{j\neq n}^{N} (i_j - 1) S_j \quad \text{and} \quad S_j = \prod_{k\neq n}^{j-1} I_k.$$

Let $d = [I_1, I_2, \cdots, I_N]$, the inverse operator of $\mathrm{unfolding}$ is denoted as $\mathrm{Folding}$ and defined as follows:

$$\mathcal{X} \ = \ \mathrm{Folding}(\mathbf{X}_{(n)}, d, n). \tag{4}$$

Definition 5. For $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ and $\mathbf{Z} \in \mathbb{R}^{K\times I_n}$ define their n-mode tensor-matrix product as a tensor $\mathcal{X} \times_n \mathbf{Z}$ of size $I_1 \times \cdots I_{n-1} \times K \times I_{n+1} \cdots \times I_N$ and with the entries

$$(\mathcal{X} \times_n \mathbf{Z})_{i_1\cdots i_{n-1} k i_{n+1}\cdots i_N} = \sum_{i_n=1}^{I_n} \mathcal{X}_{i_1\dots i_N} \mathbf{Z}_{k i_n}. \tag{5}$$

This can also be expressed in terms of unfolded tensors as follows

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{Z} \iff \mathbf{Y}_{(n)} = \mathbf{Z}\mathbf{X}_{(n)}. \tag{6}$$

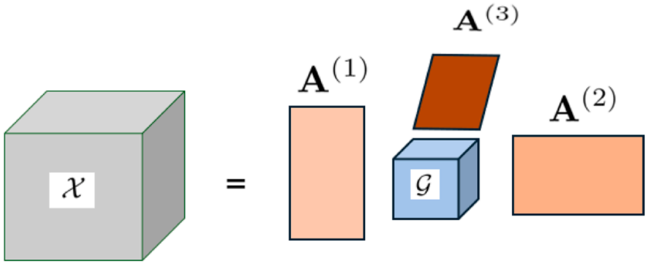


Fig. 1. Tucker decomposition for third order tensor $\mathcal{X}$.

4. The proposed gap-filling algorithms

Given an incomplete MoCap tensor, denoted as $\mathcal{T} \in \mathbb{R}^{T\times J\times 3}$, where T represents the time dimension (frames), J corresponds to the number of joints (markers), and 3 denotes the three spatial coordinates (x, y, z), the tensor captures the dynamic movement of a human subject across time and space. Gaps arise primarily due to the nature of marker-based motion capture systems. In such systems, markers are placed on specific joints or body parts and tracked using multiple cameras. However, during the recording process, several issues can lead to the problem of missing markers. For instance, markers can become occluded by other parts of the body or environment, displaced, or even fall off completely. When this occurs, the corresponding data points for those markers are missing, leading to gaps in the motion data. We approach this problem using a tensor completion framework. We consider the context of a general tensor $\mathcal{X}$ of order N, where N denotes the number of modes or dimensions of the tensor, providing a framework for dealing with the missing data problem in a multi-dimensional setting.

4.1. Tucker tensor decomposition

Tensor decomposition aims to decompose tensor data to extract features and patterns that enhance data understanding, prediction, and analysis. Tucker decomposition, in particular, factorizes multidimensional tensors into a core tensor multiplied by matrices along each mode (see Fig. 1 for the third-order tensor). For a given tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ of order N, Tucker decomposition is given as

$$\mathcal{X} = \sum_{s_1=1}^{R_1}\sum_{s_2=1}^{R_2}\dots\sum_{s_N=1}^{R_N} \mathcal{G}_{s_1\dots s_N}\mathbf{a}^{(1)}_{s_1}\circ\mathbf{a}^{(2)}_{s_2}\circ\dots\circ\mathbf{a}^{(N)}_{s_N} = \mathcal{G}\times_1\mathbf{A}^{(1)}\times_2\mathbf{A}^{(2)}\dots\times_N\mathbf{A}^{(N)} =: [\![\mathcal{G};\{\mathbf{A}\}]\!],$$

where the symbol "$\circ$" represents the vector's outer product, $\mathcal{G} \in \mathbb{R}^{R_1\times R_2\times\cdots\times R_N}$ $(R_n \leq I_n)$ is *the core tensor*, $\{\mathbf{A}\}$ stands for the set of N matrices, and $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n\times R_n}$ for $n \in [N]$ is *factor matrix*. The Tucker rank, also called the multilnear rank; is the vector $R = [R_1, R_2, \dots, R_N]$, thus we say $\mathcal{X}$ is rank-$(R_1, R_2, \dots, R_N)$ tensor. The matricized version of Tucker decomposition is given as

$$\mathbf{X}_{(n)} = \mathbf{A}^{(n)}\mathbf{G}_{(n)}(\mathbf{A}^{(N)}\otimes\dots\mathbf{A}^{(n+1)}\otimes\mathbf{A}^{(n-1)}\dots.\otimes\mathbf{A}^{(1)})^T, \tag{7}$$

where $\mathbf{G}_{(n)}$ denotes mode-n matricization of $\mathcal{G}$ and $\mathbf{X}^T$ denotes the transpose of matrix $\mathbf{X}$. Tucker decomposition is generally non-unique, so practical applications often impose constraints on the core tensor and factor matrices to ensure a meaningful factorization. These constraints, such as orthogonality, non-negativity, or sparsity, are selected based on the specific requirements of the problem. When the factor matrices are orthogonal, they capture the most significant patterns or components in the data along each tensor dimension (or mode). Meanwhile, the elements of the core tensor $\mathcal{G}$ represent the level of interaction between the different modes. It's important to note that orthogonality does not guarantee a unique decomposition. The factor matrices are only determined up to orthogonal transformations, meaning that if $(\mathcal{G}; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)})$ is a Tucker decomposition, then so is $(\mathcal{G}\times_1\mathbf{Q}^{(1)}\times_2\mathbf{Q}^{(2)}\cdots\times_N\mathbf{Q}^{(N)}; \mathbf{A}^{(1)}(\mathbf{Q}^{(1)})^T, \dots, \mathbf{A}^{(N)}(\mathbf{Q}^{(N)})^T)$ for any orthogonal matrices $\mathbf{Q}^{(n)} \in \mathbb{R}^{R_n\times R_n}$. These constraints, such as orthogonality, non-negativity, or sparsity, are selected based on the specific requirements of the problem. When the factor matrices are orthogonal, they capture the most significant patterns or components in the data along each tensor dimension (or mode). Meanwhile, the elements of the core tensor $\mathcal{G}$ represent the level of interaction between the different modes, providing a compact representation that preserves the multilinear structure of the original data. The Tucker decomposition problem of decomposing a tensor $\mathcal{X}$ can be written as the following minimization problem:

$$\min_{\mathcal{G},\mathbf{A}^{(1)},\dots,\mathbf{A}^{(N)}} \frac{1}{2}\|\mathcal{X} - \mathcal{G}\times_1\mathbf{A}^{(1)}\times_2\mathbf{A}^{(2)}\dots\times_N\mathbf{A}^{(N)}\|_F^2, \tag{8}$$

Several algorithms have been developed to solve this optimization problem. The Tucker decomposition was first introduced by Tucker in [34] and further developed in [31]. The original method was formalized as the Higher-Order Singular Value Decomposition (HOSVD) [35]. The efficient Higher-Order Orthogonal Iteration (HOOI) algorithm was proposed to find the best rank-$(R_1, R_2, \dots, R_N)$ approximation of a given tensor through iterative optimization [35]. These algorithmic developments have established Tucker decomposition as a fundamental tool in multilinear algebra with applications spanning chemometrics, signal processing, computer vision, and data mining. Alternating Least Squares (ALS) algorithms have been developed as an alternative approach for computing a Tucker decomposition of three-way arrays [36].

4.2. Tucker decomposition for gap-filling problem

We consider the Tucker decomposition with columnwise orthonormal factor matrices to tackle the gap-filling problem. This constraint ensures that the components in each mode are independent, improving the numerical stability of the decomposition. This means a set of independent features represents each mode factor. Thus, we consider the following tensor completion problem:

$$\begin{aligned} \min_{\mathbf{A},\mathcal{G},\mathcal{X}} \mathcal{L}(\mathbf{A},\mathcal{G},\mathcal{X}) \;&=\; \min_{\mathbf{A},\mathcal{G},\mathcal{X}} \frac{1}{2}\|\mathcal{X} - \mathcal{G}\times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \dots \times_N \mathbf{A}^{(N)}\|_F^2, \\ \text{such that} \qquad & \mathcal{X}_\Omega = \mathcal{T}_\Omega, \quad \mathbf{A}^{(n)}\mathbf{A}^{(n),T} = \mathbf{I}_n, \quad n \in [N], \end{aligned} \tag{9}$$

where $(.)^T$ is the transpose operator, Ω represents the set of indices for the observed elements, $\mathcal{T}_\Omega$ is the observed entries, and $\mathcal{X}_\Omega = \mathcal{T}_\Omega$ stands for

$$\mathcal{X}_{i_1,i_2,\dots,i_N} = \begin{cases} \mathcal{T}_{i_1,i_2,\dots,i_N} & \text{if } (i_1,i_2,\dots,i_N) \in \Omega, \\ 0 & \text{otherwise.} \end{cases}$$

By introducing

$$\Phi(\mathcal{X}) = \begin{cases} 0 & \text{if } \mathcal{X}_\Omega = \mathcal{T}_\Omega, \\ \infty & \text{otherwise.} \end{cases}, \quad \mathcal{R}_n(\mathbf{A}^{(n)}) = \begin{cases} 0 & \text{if } \mathbf{A}^{(n),T}\mathbf{A}^{(n)} = \mathbf{I}_n, \\ \infty & \text{otherwise.} \end{cases}, \quad \mathcal{R}(\mathbf{A}) = \sum_{n=1}^{N} \mathcal{R}_n(\mathbf{A}^{(n)}), \tag{10}$$

the completion problem can be formulated as follows

$$\min_{\mathbf{A},\mathcal{G},\mathcal{X}} \mathcal{H}(\mathbf{A},\mathcal{G},\mathcal{X}) = \min_{\mathbf{A},\mathcal{G},\mathcal{X}} \mathcal{L}(\mathbf{A},\mathcal{G},\mathcal{X}) + \Phi(\mathcal{X}) + \mathcal{R}(\mathbf{A}), \tag{11}$$

4.3. The proximal block coordinate descent (prox-BCD) algorithm

Let f be a real-valued function of s real variables $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_s)$ belonging to the product space $\mathbb{R}^{n_1} \times \dots \times \mathbb{R}^{n_s}$, where each x_i belongs to $\mathbb{R}^{n_i}$. We are interested in the minimization of functions $f : \mathbb{R}^{n_1} \times \dots \times \mathbb{R}^{n_p} \to \mathbb{R} \cup \{+\infty\}$ having the following form

$$\min_{\mathbf{x}_1,\mathbf{x}_2,\dots\mathbf{x}_s} f(\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_s) = \min_{\mathbf{x}_1,\mathbf{x}_2,\dots\mathbf{x}_s} g(\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_s) + \sum_{i=1}^{s} g_i(\mathbf{x}_i), \tag{12}$$

where g is a C^1 function with locally Lipschitz continuous gradient, and $g_i : \mathbb{R}^{n_i} \to \mathbb{R} \cup \{+\infty\}$ is a proper lower semicontinuous function, $i = 1, 2, \dots, s$. The Proximal Regularized BCD (Prox-BCD) method incorporates a proximal term into the objective function and iteratively updates one block of variables at a time [37–39].

The Prox-BCD algorithm first uses a proximal point modification of (12) as follows

$$\min_{\mathbf{x}_1,\mathbf{x}_2,\dots\mathbf{x}_s} f(\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_s) + \sum_{i=1}^{s} \frac{\rho_i}{2}\|\mathbf{x}_i - \mathbf{x}_i^k\|_F^2, \tag{13}$$

where $\frac{\rho_i}{2}\|\mathbf{x}_i - \mathbf{x}_i^k\|_F^2$ is the proximal term added to the objective to regularize the update, preventing large deviations from the current estimate $\mathbf{x}_i^k$ and ρ_i^s is a regularization parameter. Given $\mathbf{x}^0 = (\mathbf{x}_1^0, \mathbf{x}_2^0, \dots, \mathbf{x}_s^0)$ as an initial estimate, the algorithm updates the estimates of $\mathbf{x}^{k+1} = (\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \dots, \mathbf{x}_s^{k+1})$ alternately in the $(k+1)^{th}$ iteration as follows:

$$\begin{aligned} \mathbf{x}_1^{k+1} &= \arg\min_{x_1} f(\mathbf{x}_1, \mathbf{x}_2^k, \dots, \mathbf{x}_s^k) + \frac{\rho_1}{2}\|\mathbf{x}_1 - \mathbf{x}_1^k\|_F^2, \\ &\;\vdots \\ \mathbf{x}_i^{k+1} &= \arg\min_{\mathbf{x}_i} f(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \dots, \mathbf{x}_{i-1}^{k+1}, \mathbf{x}_i, \mathbf{x}_{i+1}^k, \dots, \mathbf{x}_s^k) + \frac{\rho_i}{2}\|\mathbf{x}_i - \mathbf{x}_i^k\|_F^2, \\ &\;\vdots \\ \mathbf{x}_s^{k+1} &= \arg\min_{\mathbf{x}_s} f(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \dots, \mathbf{x}_s) + \frac{\rho_s}{2}\|\mathbf{x}_s - \mathbf{x}_s^k\|_F^2. \end{aligned} \tag{14}$$

4.4. The minimization algorithm

We consider the Prox-BCD algorithm (14) to minimize the objective function in (11). Each of the variables $\mathbf{A}, \mathcal{G}, \mathcal{X}$ is regularized with a quadratic proximal term. These terms enforce that the updated estimates for the variables do not deviate significantly from their current values, which are denoted $\mathbf{A}_k, \mathcal{G}_k, \mathcal{X}_k$ at the k^{th} iteration. The regularization parameters ρ_a, ρ_g, ρ_x control the influence of these penalty terms. Thus, we obtain the following minimization problem

$$\min_{\mathbf{A},\mathcal{G},\mathcal{X}} \mathcal{H}(\mathbf{A},\mathcal{G},\mathcal{X}) + \frac{\rho_a}{2}\sum_{n=1}^{N}\|\mathbf{A}^{(n)} - \mathbf{A}_k^{(n)}\|_F^2 + \frac{\rho_g}{2}\|\mathcal{G} - \mathcal{G}_k\|_F^2 + \frac{\rho_x}{2}\|\mathcal{X} - \mathcal{X}_k\|_F^2. \tag{15}$$

This approach involves iteratively optimizing one block of variables while keeping the others fixed. Thus, the iterative minimization algorithm involves three main steps: factor matrices (16a), the core tensor (16b), and tensor recovery (16c). Given the $(\{\mathbf{A}_0^{(n)}\}_{n=1}^N, \mathcal{G}_0, \mathcal{X}_0)$ as the initial estimate, in the $(k+1)^{th}$ iteration, we have

$$\mathbf{A}_{k+1}^{(n)} = \operatorname{argmin}_{\mathbf{A}^{(n)}} \mathcal{H}(\mathbf{A}, \mathcal{G}_k, \mathcal{X}_k) + \frac{\rho_a}{2} \|\mathbf{A}^{(n)} - \mathbf{A}_k^{(n)}\|_F^2, \quad \text{For} \quad n \in [N] \tag{16a}$$

$$\mathcal{G}_{k+1} = \operatorname{argmin}_{\mathcal{G}} \mathcal{H}(\mathbf{A}_{k+1}, \mathcal{G}, \mathcal{X}_k) + \frac{\rho_g}{2} \|\mathcal{G} - \mathcal{G}_k\|_F^2, \tag{16b}$$

$$\mathcal{X}_{k+1} = \operatorname{argmin}_{\mathcal{X}} \mathcal{H}(\mathbf{A}_{k+1}, \mathcal{G}_{k+1}, \mathcal{X}) + \frac{\rho_x}{2} \|\mathcal{X} - \mathcal{X}_k\|_F^2. \tag{16c}$$

4.4.1. The factor matrices updates

The objective function of the sub-problems $\mathbf{A}_{k+1}^{(n)}$, $\forall n \in [N]$ in (16a), can be solved simultaneously by decomposing them into N independent minimization problems. For $n \in [N]$, the factor matrices $\mathbf{A}^{(n)}$ are updated using the matricized form (7). Thus we obtain

$$\mathbf{A}_{k+1}^{(n)} = \arg\min_{\mathbf{A}^{(n)}} \mathcal{F}_n(\mathbf{A}^{(n)}) = \arg\min_{\mathbf{A}^{(n)}} \ \frac{1}{2} \|\mathbf{X}_{(n),k} - \mathbf{A}^{(n)} \mathbf{G}_{(n),k} \{ \bigotimes_{i \neq n}^{i=N} \mathbf{A}^{(i)} \}^T \|_F^2 + \frac{\rho_a}{2} \|\mathbf{A}^{(n)} - \mathbf{A}_k^{(n)}\|_F^2 + \mathcal{R}_n(\mathbf{A}^{(n)}),$$

where $\{\bigotimes_{i \neq n}^{i=N} \mathbf{A}^{(i)}\} = \mathbf{A}_k^{(N)} \otimes \dots \mathbf{A}_k^{(n+1)} \otimes \mathbf{A}_{k+1}^{(n-1)} \dots . \otimes \mathbf{A}_{k+1}^{(1)}$ and $\mathbf{G}_{(n),k}$ is the mode-n matricization of $\mathcal{G}_k$. For $n \in [N]$, the partial derivative of the objective function $\mathcal{F}_n$ is

$$\nabla \mathcal{F}_n(\mathbf{A}^{(n)}) = \mathbf{A}_{k+1}^{(n)} \mathbf{G}_{(n),k} \mathbf{G}_{(n),k}^T + \rho_a \mathbf{A}_{k+1}^{(n)} + \mathbf{Z}_n \mathbf{G}_{(n),k}^T + \rho_a \mathbf{A}_k^{(n)},$$

where $\mathbf{Z}_n$ is the mode-n matricization of $\mathcal{X}_k \times_1 \mathbf{A}_{k+1}^{(1),T} \dots \times_{n-1} \mathbf{A}_{k+1}^{(n-1),T} \times_{n+1} \mathbf{A}_k^{(n+1),T} \dots \times_N \mathbf{A}_k^{(N),T}$. Thus, the updates are given as follows

$$\mathbf{A}_{k+1}^{(n)} = \left(\mathbf{Z}_n \mathbf{G}_{(n),k}^T + \rho_a \mathbf{A}_k^{(n)} \right) \left(\mathbf{G}_{(n),k} \mathbf{G}_{(n)}^T + \alpha_n \mathbf{I}_n + \rho_a \mathbf{I}_n \right)^{\dagger}, \tag{17}$$

where $\mathbf{X}^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of $\mathbf{X}$. For the columnwise orthonormality of $\mathbf{A}_{k+1}^{(n)}$, we consider the QR decomposition, which decomposes a matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ into a matrix with orthonormal columns $\mathbf{Q} \in \mathbb{R}^{I \times J}$ and an upper triangular matrix $\mathbf{R} \in \mathbb{R}^{J \times J}$. The updated matrix $\mathbf{A}_{k+1}^{(n)}$ is set to the matrix $\mathbf{Q}$, which has orthonormal columns.

4.4.2. The core tensor update

The core tensor problem can be written as

$$\mathcal{G}_{k+1} = \arg\min_{\mathcal{G}} \frac{1}{2} \|\mathcal{G} - \mathcal{X}_k \times_1 \mathbf{A}_{k+1}^{(1),T} \times_2 \mathbf{A}_{k+1}^{(2),T} \dots \times_N \mathbf{A}_{k+1}^{(N),T}\|_F^2 + \frac{\rho_g}{2} \|\mathcal{G} - \mathcal{G}_k\|_F^2.$$

Setting the partial derivative of the objective function with respect to $\mathcal{G}$ to zeros, we obtain

$$\mathcal{G}_{k+1} - \mathcal{X}_k \times_1 \mathbf{A}_{k+1}^{(1),T} \times_2 \mathbf{A}_{k+1}^{(2),T} \dots \times_N \mathbf{A}_{k+1}^{(N),T} + \rho_g \mathcal{G}_{k+1} - \rho_g \mathcal{G}_k = 0.$$

The core tensor is then updated as

$$\mathcal{G}_{k+1} = \frac{1}{1 + \rho_g} \Big(\mathcal{X}_k \times_1 \mathbf{A}_{k+1}^{(1),T} \times_2 \mathbf{A}_{k+1}^{(2),T} \dots \times_N \mathbf{A}_{k+1}^{(N),T} + \mathcal{G}_k \Big). \tag{18}$$

4.4.3. Tensor recovery step

For the tensor recovery, we solve the following minimization problem:

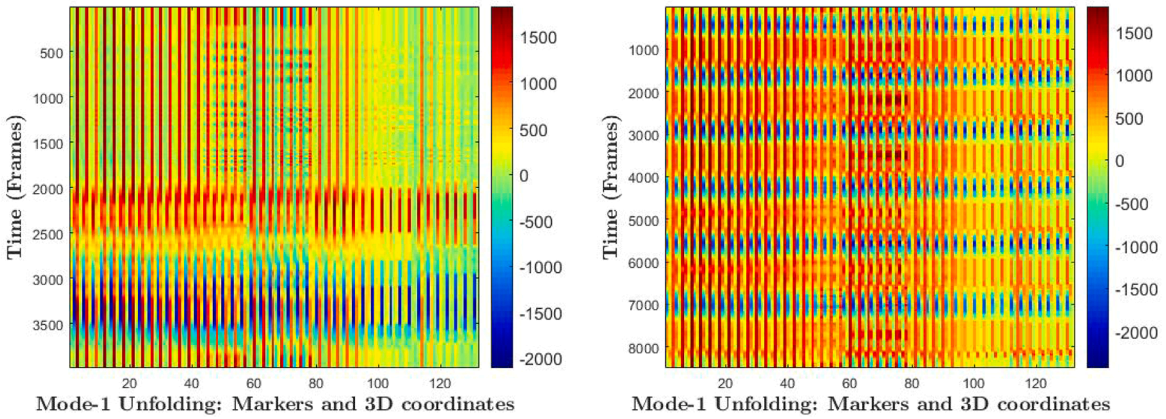
$$\mathcal{X}_{k+1} = \arg\min_{\mathcal{X}} \ \frac{1}{2} \|\mathcal{X} - \mathcal{G}_{k+1} \times_1 \mathbf{A}_{k+1}^{(1)} \times_2 \mathbf{A}_{k+1}^{(2)} \dots \times_N \mathbf{A}_{k+1}^{(N)}\|_F^2 + \frac{\rho_x}{2} \|\mathcal{X} - \mathcal{X}_k\|_F^2 + \Phi(\mathcal{X}).$$

For each iteration, since $\mathcal{X}_\Omega = \mathcal{T}_\Omega$, we update the unobserved entries (i.e., at the positions in Ω^c , the complement of Ω) using the current estimates of the core tensor $\mathcal{G}_{k+1}$ and the factor matrices $\mathbf{A}_{k+1}^{(n)}$

$$(\mathcal{X}_{k+1})_{\Omega_c} = \frac{\mathcal{G}_{k+1} \times_1 \mathbf{A}_{k+1}^{(1)} \dots \times_N \mathbf{A}_{k+1}^{(N)} + \rho_x \mathcal{X}_k}{1 + \rho_x},$$

where $(\mathcal{X}_{k+1})_{\Omega_c}$ is the recovered tensor for the unobserved entries. The final update for $\mathcal{X}_{k+1}$ is the projection of the reconstructed tensor onto the unobserved entries set Ω^c and the observed entries $\mathcal{T}_\Omega$

$$\mathcal{X}_{k+1} = \left(\frac{\mathcal{G}_{k+1} \times_1 \mathbf{A}_{k+1}^{(1)} \dots \times_N \mathbf{A}_{k+1}^{(N)} + \rho_x \mathcal{X}_k}{1 + \rho_x} \right)_{\Omega^c} + \mathcal{T}_\Omega. \tag{19}$$



(a) (b)

Fig. 2. Temporal mode-1 unfolding, $\mathbf{T}_{(1)}$, of the MoCap tensor $\mathcal{T}$ for the HDM1 and HDM4 sequences.

4.5. *Tucker decomposition with temporal nuclear norm for gap-filling problem*

Time series data analysis, which inherently exhibits temporal dependencies, often requires models that capture the evolving structure over time. In particular, MoCap data possesses temporal continuity as the motion changes smoothly over time. For instance, Fig. 2 illustrates the temporal mode of two MoCap sequences, showing the evolution of joint positions over time. We can observe how the data evolves smoothly with gradual transitions in the marker positions. We can observe clear periodicity in the motion, with repetitive patterns emphasizing the low-rank structure and temporal smoothness inherent in MoCap data. The smoothness of the transitions between frames indicates that the motion evolves gradually over time, maintaining continuity and consistency. Therefore, we consider the nuclear norm of the temporal unfolding mode of the MoCap tensor. The nuclear norm promotes low-rank temporal structure, encouraging smoothness by representing motion with fewer temporal components while naturally filtering out noise and abrupt transitions. The temporal nuclear norm enforces global smoothness via rank minimization rather than costly local pairwise continuity. Incorporating the temporal nuclear norm alongside Tucker decomposition helps reduce abrupt variations, ensuring that the reconstructed values follow coherent and consistent temporal patterns.

We propose the following Tucker-based temporal nuclear norm (TuckerTNN) approach

$$\begin{aligned} &\min_{\mathbf{A},\mathcal{G},\mathcal{X}} \frac{1}{2}\|\mathcal{X}-\mathcal{G}\times_1 \mathbf{A}^{(1)}\times_2 \mathbf{A}^{(2)}\dots\times_N \mathbf{A}^{(N)}\|_F^2 + \lambda\|\mathbf{X}_{(1)}\|_*, \\ &\quad \text{s.t} \quad \mathcal{X}_\Omega = \mathcal{T}_\Omega, \quad \mathbf{A}^{(n)}\mathbf{A}^{(n),T} = \mathbf{I}_n, \quad n\in[N]. \end{aligned} \tag{20}$$

Analog to (9), we consider the indicator functions (10) and use the Prox-BCD algorithm (14). Thus we obtain

$$\min_{\mathbf{A},\mathcal{G},\mathcal{X}} \hat{\mathcal{H}}(\mathbf{A},\mathcal{G},\mathcal{X}) + \frac{\rho_a}{2}\|\mathbf{A}-\mathbf{A}_k\|_F^2 + \frac{\rho_g}{2}\|\mathcal{G}-\mathcal{G}_k\|_F^2 + \frac{\rho_x}{2}\|\mathcal{X}-\mathcal{X}_k\|_F^2, \tag{21}$$

where

$$\hat{\mathcal{H}}(\mathbf{A},\mathcal{G},\mathcal{X}) = \frac{1}{2}\|\mathcal{X}-\mathcal{G}\times_1 \mathbf{A}^{(1)}\times_2 \mathbf{A}^{(2)}\dots\times_N \mathbf{A}^{(N)}\|_F^2 + \mathcal{R}(\mathbf{A}) + \lambda\|\mathbf{X}_{(1)}\|_* + \Phi(\mathcal{X}). \tag{22}$$

The iterative minimization algorithm (14) involves three main steps: factor matrices (23a), the core tensor (23b), and tensor recovery (23c):

$$\mathbf{A}_{k+1} = \operatorname{argmin}_{\mathbf{A}} \hat{\mathcal{H}}(\mathbf{A},\mathcal{G}_k,\mathcal{X}_k) + \frac{\rho_a}{2}\|\mathbf{A}-\mathbf{A}_k\|_F^2, \tag{23a}$$

$$\mathcal{G}_{k+1} = \operatorname{argmin}_{\mathcal{G}} \hat{\mathcal{H}}(\mathbf{A}_{k+1},\mathcal{G},\mathcal{X}_k) + \frac{\rho_g}{2}\|\mathcal{G}-\mathcal{G}_k\|_F^2, \tag{23b}$$

$$\mathcal{X}_{k+1} = \operatorname{argmin}_{\mathcal{X}} \hat{\mathcal{H}}(\mathbf{A}_{k+1},\mathcal{G}_{k+1},\mathcal{X}) + \frac{\rho_x}{2}\|\mathcal{X}-\mathcal{X}_k\|_F^2. \tag{23c}$$

The first and the second steps remain the same, only the tensor recovery step changes.

4.5.1. *Tensor recovery*

This step is mainly based on entailing the solution to the following matrix nuclear norm minimization problem

$$\mathbf{X}_{(1),k+1} = \arg\min_{\mathbf{X}_{(1)}} \frac{1}{2}\|\mathbf{X}_{(1)} - \mathbf{A}_{k+1}^{(1)}\mathbf{G}_{(1),k+1}\{\bigotimes_{n\neq 1}^{n=N}\mathbf{A}_{k+1}^{(n)}\}^T\|_F^2 + \frac{\rho_x}{2}\|\mathbf{X}_{(1)}-\mathbf{X}_{(1),k}\|_F^2 + \lambda\|\mathbf{X}_{(1)}\|_*. \tag{24}$$

To address problem (24), we consider an Accelerated Proximal Gradient (APG) method.

4.5.2. Accelerated proximal gradient (APG) algorithm

The APG methods are iterative optimization algorithms designed to solve non-smooth convex problems. These methods combine proximal gradient descent with acceleration techniques, such as Nesterov's acceleration [40], to achieve faster convergence. At each iteration, APG methods update the solution by incorporating the gradient of the smooth part of the objective and applying a proximal operator for the nonsmooth term. The acceleration speeds up convergence compared to standard gradient-based methods, especially for large-scale or structured problems. Instead of making quadratic approximations around the point $\mathbf{X}^p$, APG makes the approximation at another point $\mathbf{Y}^p$ which is a linear combination of $\mathbf{X}^p$. This simple modification will give a convergence rate of $O(1/p^2)$, where p is the number of iterations. To update the $\mathbf{X}_{(1),k+1}$ in (24), we adopt the APG algorithm proposed in [41] which solves problems of the following form:

$$\min_{\mathbf{X}} \; h(\mathbf{X}) + g(\mathbf{X}), \tag{25}$$

where h is a convex and differentiable function and g is a closed, convex, possibly nondifferentiable function. First, for any $p > 0$, the APG method which has the unique minimizer $\hat{\mathbf{X}}$. The APG solves the optimization problem (25) by iteratively updating $\mathbf{X}, \mathbf{Y}$ and t. Given $(\mathbf{X}^1, \mathbf{Y}^1, t^1)$, at the $(p+1)^{th}$ iteration, we update the sequence $(\mathbf{X}^{p+1}, \mathbf{Y}^{p+1}, t^{p+1})$ as follows

$$\mathbf{X}^{p+1} = \arg\min_{\mathbf{X}} \; q(\mathbf{X}, \mathbf{Y}^p) = \arg\min_{\mathbf{X}} g(\mathbf{X}) + \frac{L}{2}\left\| \mathbf{X} - \left(\mathbf{Y}^p - \frac{1}{L}\nabla h(\mathbf{Y}^p) \right) \right\|_F^2, \tag{26a}$$

$$t^{p+1} = \frac{1 + \sqrt{1 + 4t^{p2}}}{2}, \tag{26b}$$

$$\mathbf{Y}^{p+1} = \mathbf{X}^{p+1} + \frac{t^p - 1}{t^{p+1}}\left(\mathbf{X}^{p+1} - \mathbf{X}^p \right). \tag{26c}$$

We first introduce the following useful tool, the singular values shrinkage operator, to solve the nuclear norm problem (24).

Definition 6. We Consider the SVD of a matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$, $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. We define the singular values shrinkage operator $\mathcal{D}_\lambda$ as follows:

$$\mathcal{D}_\lambda(\mathbf{X}) = \mathbf{U}\mathcal{S}_\lambda(\boldsymbol{\Sigma})\mathbf{V}^T,$$

where $\mathcal{S}_\lambda(\boldsymbol{\Sigma}) = \operatorname{diag}\left(\max\left\{ \sigma_i - \lambda, 0 \right\} \right)$ is the soft-thresholding operator of $\boldsymbol{\Sigma}$.

Thus, we have the following useful theorem which proves that the shrinkage operator is the proximal mapping associated with the nuclear norm (see [42] for more details).

Theorem 1 ([42]). *For each $\lambda > 0$ and $\mathbf{Y} \in \mathbb{R}^{I \times J}$, we have*

$$\mathcal{D}_\lambda(\mathbf{Y}) = \arg\min_{\mathbf{X}} \frac{1}{2}\|\mathbf{X} - \mathbf{Y}\|_F^2 + \lambda\|\mathbf{X}\|_*. \tag{27}$$

We describe the APG-based singular value thresholding (APG-SVT) algorithm for solving (24). We consider $\mathbf{Z}_{k+1} = \mathbf{A}_{k+1}^{(1)}\mathbf{G}_{(1),k+1}\{\bigotimes_{n \neq 1}^{n=N} \mathbf{A}_{k+1}^{(n)}\}^T$, we have

$$\begin{aligned} h(\mathbf{X}_{(1)}) &= \frac{1}{2}\|\mathbf{X}_{(1)} - \mathbf{Z}_{k+1}\|_F^2 + \frac{\rho_x}{2}\|\mathbf{X}_{(1)} - \mathbf{X}_{(1),k}\|_F^2, \\ g(\mathbf{X}_{(1)}) &= \lambda\|\mathbf{X}_{(1)}\|_*. \end{aligned} \tag{28}$$

According to the APG algorithm, we have

$$\begin{aligned} \mathbf{X}_{(1),k+1}^{p+1} &= \arg\min_{\mathbf{X}_{(1)}} h(\mathbf{X}_{(1)}) + \lambda\|\mathbf{X}_{(1)}\|_*, \\ &= \arg\min_{\mathbf{X}_{(1)}} \frac{L}{2}\left\| \mathbf{X}_{(1)} - \left(\mathbf{Y}^p - \frac{1}{L}\nabla h(\mathbf{Y}^p) \right) \right\|_F^2 + \lambda\|\mathbf{X}_{(1)}\|_*, \\ &= \mathcal{D}_{\lambda/L}(\mathbf{Y}^p - \frac{1}{L}\nabla h(\mathbf{Y}^p)), \end{aligned} \tag{29}$$

where $\nabla h(\mathbf{X}) = (1 + \rho_x)\mathbf{X} - (\mathbf{Z}_{k+1} + \rho_x\mathbf{X}_{(1),k})$. $\mathbf{Y}^{p+1}$ and t^{p+1} are updated by (26c) and (26b). We summarize the process of APG-SVT in Algorithm 1.

After obtaining the temporal mode $\mathbf{X}_{(1)}$ from Algorithm 1, let $d = [T, J, 3]$, we update the tensor $\mathcal{X}_{k+1}$ as follows:

$$\mathcal{X}_{k+1} = \left(\text{Folding}(\mathbf{X}_{(1),k+1}, d, 1) \right)_{\Omega^c} + \mathcal{T}_\Omega. \tag{30}$$

5. Rank-increasing scheme for rank estimation

A key challenge in tensor decompositions is determining the decomposition ranks, which are often unknown, particularly in tensor completion problems. To estimate the ranks for the Tucker decomposition, we implement a rank-increasing scheme that begins with underestimated ranks and progressively increases them if the algorithm exhibits slow progress. The initial rank is set to

Algorithm 1 APG-SVT algorithm.

```
Require: $\lambda > 0$, $L > 0$.
Initialize $\mathbf{X}^1_{(1),k+1} \in \mathbb{R}^{T\times 3J}$, $\mathbf{Y}^1, t^0 = t^1 = 1$.
for $p = 1, \dots, P$ do
    update $\mathbf{B}^p = \mathbf{Y}^p - \frac{1}{L}\nabla h(\mathbf{Y}^p)$,
    compute $\mathcal{D}_{\lambda/L}(\mathbf{B}^p)$ from the SVD of $\mathbf{B}^p$,
    update $\mathbf{X}^{p+1}_{(1),k+1} = \mathcal{D}_{\lambda/L}(\mathbf{B}^p)$,
    update $\mathbf{Y}^{p+1} = \mathbf{X}^{p+1}_{(1),k+1} + \frac{t^{p-1}-1}{t^{p+1}}\left(\mathbf{X}^{p+1}_{(1),k+1} - \mathbf{X}^{p}_{(1),k+1}\right)$,
    Compute $t^{p+1} = \frac{1+\sqrt{1+4(t^p)^2}}{2}$.
end for
Return $\mathbf{X}_{(1),k+1}$.
```

Table 1
Motion sequences from the HDM05 dataset used in the experiments.

Motion	File name	Description	Frames	Markers	Frequency
HDM1	HDM_mm_01-02_03_120	Locomotion on the spot	3990	44	120
HDM2	HDM_bd_05-01_01_120	Clapping and waving	5920	44	120
HDM3	HDM_mm_03-02_01_120	Kicking and punching	7671	44	120
HDM4	HDM_mm_02-02_02_120	Shelf (while walking)	8500	44	120

$R = [R_1, R_2, \dots, R_N]$ such that $R_n \le R_n^{\max}$. The criterion for increasing the Tucker decomposition rank is based on the relative change in reconstruction error, mathematically expressed as:

$$\left| 1 - \frac{\|(\mathcal{G}_{k+1} \times_1 \mathbf{A}^{(1)}_{k+1} \dots \times_N \mathbf{A}^{(N)}_{k+1})_\Omega - \mathcal{T}_\Omega\|_F}{\|(\mathcal{G}_{k} \times_1 \mathbf{A}^{(1)}_{k} \dots \times_N \mathbf{A}^{(N)}_{k})_\Omega - \mathcal{T}_\Omega\|_F} \right| \le \epsilon, \tag{31}$$

where $\mathcal{T}_\Omega$ is the observed entries and ϵ is a tolerance for the relative change in the reconstruction error. If this condition is met, it indicates minimal improvement in reconstruction accuracy. When this condition is met, the Tucker rank is increased to $R_n + \Delta R_n$ at the $(k+1)^{th}$ iteration, where ΔR_n is a positive integer. When increasing the rank R_n we update the factor matrices as follows

$$\begin{aligned}
&\mathbf{A}^{(n)}_{k+1} = [\mathbf{A}^{(n)}_{k+1}, \mathrm{randn}(I_n, \Delta R_n)],\ n \in [N]\\
&[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{A}^{(n)}_{k+1}),\ n \in [N]\\
&\mathbf{A}^{(n)}_{k+1} = \mathbf{Q},\ n \in [N]\\
&\mathcal{G}_{k+1} = \mathcal{X}_k \times_1 \mathbf{A}^{(1),T}_{k+1} \times_2 \mathbf{A}^{(2),T}_{k+1} \dots \times_N \mathbf{A}^{(N),T}_{k+1},
\end{aligned} \tag{32}$$

where randn is a MATLAB function for generating random values.

6. Experiments

This section presents the experimental results evaluating the performance of the two proposed algorithms for addressing the gap-recovery problem in MoCap systems. We define the experimental setup, specifying the MoCap data, gap settings, and evaluation criteria. The proposed algorithms are compared with two recent CP-based gap-filling methods [20] for MoCap data. We analyze the results of multiple tests performed on a variety of motion sequences. The experiments were conducted using the MoCap toolbox[2] in MATLAB (R2018b).

MoCap data: In our experiments, we utilize the motion sequences from the online motion capture database HDM05[3] [43]. This database includes well-documented motion capture data in the C3D and the ASF/AMC formats. In Table 1 we present four MoCap datasets consisting of a variety of human motions, such as locomotion (HDM1), clapping and waving (HDM2), kicking and punching (HDM3), and shelf handling while walking (HDM4). Each dataset consists of 44 markers, recorded at a frequency of 120 Hz. The number of frames varies across the files, ranging from 3990 in HDM1 to 8500 in HDM4.

Gap setting: We simulate real-world complexities by introducing gaps into motion sequences. These gaps were generated by randomly removing markers at varying positions and durations within the sequences. Further, we take into account the dynamic nature of marker visibility in practical scenarios, where markers may disappear intermittently and reappear later during motion capture. Therefore, we consider the case where multiple gaps occur in a marker trajectory for validation purposes, as illustrated in Fig. 3. The number of missing frames and the duration of gaps are dependent. We can either fix the gap length or the gap duration. For instance, setting a

[2] https://github.com/mocaptoolbox

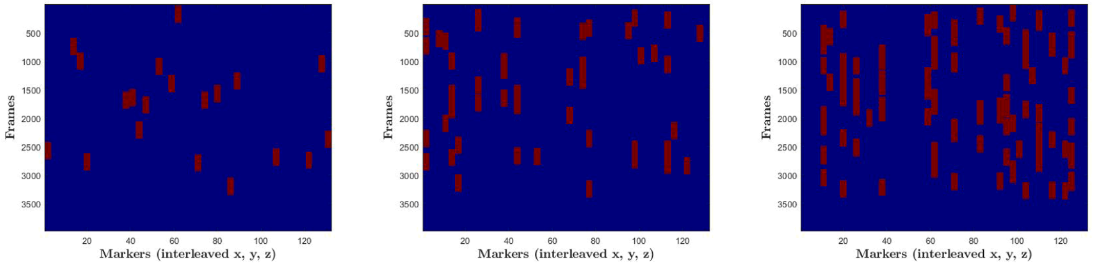
[3] https://resources.mpi-inf.mpg.de/HDM05/index.{html}

Algorithm 2 Tucker-based algorithms for the gap-filling problem.

```
Require: $\Omega$, $\mathcal{T}_\Omega$, $\{R_n\}_{n=1}^N$, $\{\Delta R_n\}_{n=1}^N$, $K$, $\rho_a$, $\rho_g$, $\rho_x$, $\epsilon$.
Randomly initialize the factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, for $n \in [N]$, and the core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$.
Set the initial known values: $\mathcal{X}_\Omega = \mathcal{T}_\Omega$.
for $k = 1, \dots, K$ do
    Step 1: Tucker decomposition
    1.1 Update the factor matrices
    for $n = 1, \dots, N$ do
        $\mathbf{A}_{k+1}^{(n)} = \left(\mathbf{Z}_n \mathbf{G}_{(n),k}^T + \rho_a \mathbf{A}_k^{(n)}\right)\left(\mathbf{G}_{(n),k}\mathbf{G}_{(n)}^T + \alpha_n \mathbf{I}_n + \rho_a \mathbf{I}_n\right)^\dagger$,
        $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{A}_{k+1}^{(n)})$,
        $\mathbf{A}_{k+1}^{(n)} = \mathbf{Q}$.
    end for
    1.2 Update the core tensor
        $\mathcal{G}_{k+1} = \frac{1}{1+\rho_g}\left(\mathcal{X}_k \times_1 \mathbf{A}_{k+1}^{(1),T} \times_2 \mathbf{A}_{k+1}^{(2),T} \dots \times_N \mathbf{A}_{k+1}^{(N),T} + \mathcal{G}_k\right)$.
    Step 2: Tensor recovery: Choose the algorithm
    a. Tucker algorithm:
        $\mathcal{X}_{k+1} = \left(\frac{\mathcal{G}_{k+1} \times_1 \mathbf{A}_{k+1}^{(1)} \dots \times_N \mathbf{A}_{k+1}^{(N)} + \rho_x \mathcal{X}_k}{1+\rho_x}\right)_{\Omega^c} + \mathcal{T}_\Omega$.
    b. TuckerTNN algorithm:
    1. Reconstruct the temporal unfolding mode $\mathbf{X}_{(1),k+1}$ by Algorithm 1,
    2. Reconstruct the tensor
        $\mathcal{X}_{k+1} = \left(\mathrm{Folding}(\mathbf{X}_{(1),k+1}, d, 1)\right)_{\Omega^c} + \mathcal{T}_\Omega$.
    Step 3: Rank-increasing
    if $\left|1 - \frac{\|(\mathcal{G}_{k+1} \times_1 \mathbf{A}_{k+1}^{(1)} \dots \times_N \mathbf{A}_{k+1}^{(N)})_\Omega - \mathcal{T}_\Omega\|_F}{\|(\mathcal{G}_k \times_1 \mathbf{A}_k^{(1)} \dots \times_N \mathbf{A}_k^{(N)})_\Omega - \mathcal{T}_\Omega\|_F}\right| \le \epsilon$, then
        for $n = 1, \dots, N$ do
            if $R_n < R_n^{\max}$ then
                Increase rank: $R_n \leftarrow R_n + \Delta R_n$,
                Update $\mathbf{A}_{k+1}^{(n)}$ with increased rank.
                    $\mathbf{A}_{k+1}^{(n)} = [\mathbf{A}_{k+1}^{(n)}, \mathrm{randn}(I_n, \Delta R_n)]$,
                    $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{A}_{k+1}^{(n)})$,
                    $\mathbf{A}_{k+1}^{(n)} = \mathbf{Q}$.
            end if
        end for
        Update core tensor $\mathcal{G}_{k+1}$ with increased ranks.
            $\mathcal{G}_{k+1} = \mathcal{X}_k \times_1 \mathbf{A}_{k+1}^{(1),T} \times_2 \mathbf{A}_{k+1}^{(2),T} \dots \times_N \mathbf{A}_{k+1}^{(N),T}$.
    end if
end for
Return $\mathcal{X}, \mathcal{G}, \{\mathbf{A}^{(n)}\}_{n=1}^N$.
```

fixed gap duration (dt) determines the total number of missing frames (M_f) based on the data frequency (F) using $M_f = dt \times F$. The missing markers are easily identifiable since they are encoded as Not a Number (NaN) entries. To rigorously evaluate gap-recovery algorithms, we investigate four key parameters:

- *Number of missing markers:* The total number of missing markers within a sequence.
- *Gap length:* The number of sequential missing frames per marker trajectory.
- *Gap duration:* The length of time during which markers are absent within a single gap.
- *Multiplicity of gaps:* The frequency of co-occurring gaps per marker trajectory.



(a)

(b)

(c)

Fig. 3. Examples of the distribution of gaps in MoCap sequences in terms of the multiplicity of gaps: (a) 1 gap per marker, (b) up to 5 gaps per marker, and (c) up to 10 gaps per marker, each gap is of 2.5 seconds duration, with 20 missing markers (HDM1).

Evaluation Criteria: The performance of each algorithm is evaluated using three key metrics: Root Mean Square Error (RMSE), Average Error (AvE), and Standard Deviation (STD), which quantify accuracy and variability. Let $\mathcal{X}$ and $\mathcal{S}$ be the recovered and the ground truth MoCap tensors, respectively, and $D = T \times J \times 3$, we have

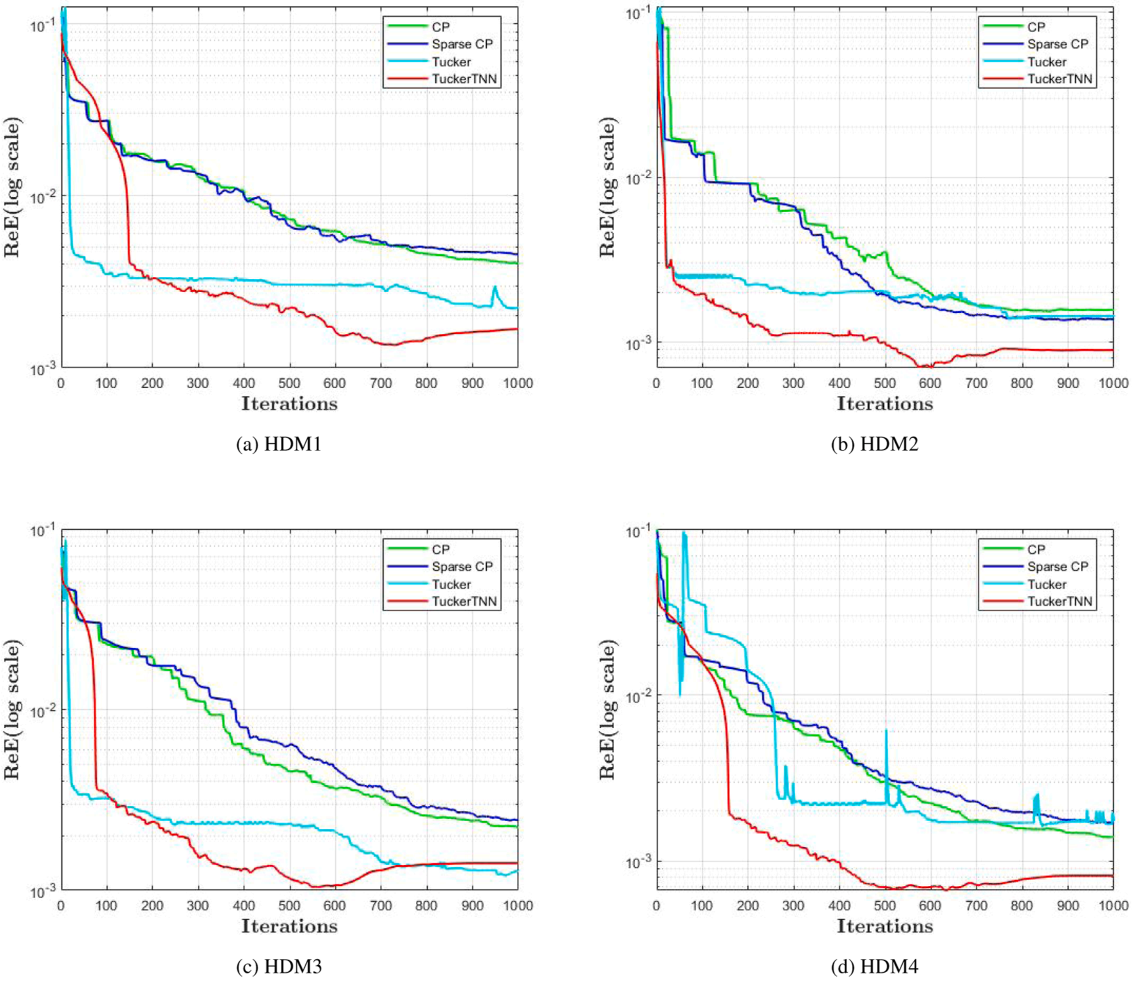
$$\text{RMSE} = \sqrt{\frac{1}{D}\sum_{t,j,k=1}^{T,J,3} |\mathcal{X}_{tjk} - \mathcal{S}_{tjk}|^2}, \qquad \text{AvE} = \frac{1}{D}\sum_{t,j,k=1}^{T,J,3} |\mathcal{X}_{tjk} - \mathcal{S}_{tjk}|,$$

$$\text{STD} = \sqrt{\frac{1}{D}\sum_{t,j,k=1}^{T,J,3} \big((\mathcal{X}_{tjk} - \mathcal{S}_{tjk}) - e\big)^2}, \quad \text{where} \ \ e = \frac{1}{D}\sum_{t,j,k=1}^{T,J,3} (\mathcal{X}_{tjk} - \mathcal{S}_{tjk}),$$

Fig. 4 illustrates four plots of the relative error (ReE) vs. iterations for CP, Sparse CP, Tucker, and TuckerTNN completion methods for the four MoCap sequences. While CP and Sparse CP show rapid initial decreases in ReE, they plateau at higher error levels. On the other hand, Tucker provides steady convergence and lower RMSE than CP and Sparse CP methods. TuckerTNN outperforms all the completion methods with the fastest decrease in ReE and the lowest error.

Table (2) provides the quantitative comparison of the four gap-filling methods-CP, SparseCP, Tucker, and TuckerTNN-evaluating their performance across different MoCap data sets and gap lengths. The methods are assessed based on RMSE, AvE, and STD, demonstrating their effectiveness under various gap scenarios. In this experiment, we set the number of missing markers to 20 while varying gap lengths (100, 150, and 200) and possible gaps per marker (2, 5, and 10). From the obtained results, we observe that Tucker and TuckerTNN outperform both CP and SparseCP across all MoCap data, demonstrating consistently lower RMSE, AvE, and STD values. When comparing the CP and Tucker decompositions, it is evident that the Tucker method provides superior performance. This highlights the effectiveness of the Tucker decomposition in capturing the complexity of MoCap data, underscoring its suitability for handling the multidimensional nature of MoCap recordings in particularly in the presence of gaps. Furthermore, the TuckerTNN algorithm consistently achieves the highest accuracy compared to the Tucker decomposition, emphasizing the benefits of the temporal nuclear norm over the Tucker method. For instance, in HDM1 with at most 2 gaps of 100-length, TuckerTNN achieves an RMSE of 0.9023, significantly lower than CP (2.558), SparseCP (2.349), and Tucker (1.577). This trend continues as gap length increases, indicating that TuckerTNN handles larger gaps more accurately. All methods exhibit increased error metrics as gap lengths grow, particularly CP and SparseCP. For example, CP's RMSE in HDM1 with almost 10 gaps rises from 4.817 at a 100-length gap to 8.338 at a 200-length gap. TuckerTNN, however, shows a more moderate increase, highlighting its robustness to longer gaps. Moreover, TuckerTNN consistently achieves the lowest STD values across all settings, indicating less variability in its recovered values and greater stability. Tucker occasionally approaches TuckerTNN's performance on data with shorter gaps. For instance, in HDM4 with at most 2 gaps and a 100-length gap, Tucker achieves an RMSE of 4.966e-1, with TuckerTNN reaching 4.465e-1. However, TuckerTNN maintains the lowest error rates overall.

Figs. 6, 7, 8, and 9 display the recovered trajectories (3D positions over time) of the missing markers for the four datasets: HDM1, HDM2, HDM3, and HDM4, respectively. In this experiment, 15 markers were randomly missing, with each gap lasting 2 seconds (equivalent to 240 frames), and each marker had up to 5 gaps. In each figure, plot (a) presents the ground truth trajectories of the selected markers, while plot (b) highlights the incomplete trajectory of each marker. We highlight inaccurately recovered markers using black arrows. Among all the methods, the CP algorithm demonstrates inaccuracy in reconstructing trajectories across all datasets. For instance, in Fig. 6, the reconstructed trajectories of markers 4 and 7 are inaccurately recovered by both the CP and SparseCP algorithms, as indicated by the arrows. In contrast, the Tucker and TuckerTNN methods provide more accurate reconstructions of the missing trajectories. In Fig. 8, the reconstructed trajectory of marker 36 clearly shows that TuckerTNN outperforms the other methods, with CP and SparseCP producing noticeably poorer results. Similarly, in Fig. 9, Marker 25 is poorly reconstructed by Tucker, whereas TuckerTNN offers a significantly more accurate recovery. When comparing CP to Tucker decomposition, the latter consistently provides accurate results, further demonstrating the ability of Tucker decomposition to handle gaps in motion capture sequences.



(a) HDM1

(b) HDM2

(c) HDM3

(d) HDM4

Fig. 4. The relative error obtained by the four algorithms versus iterations.

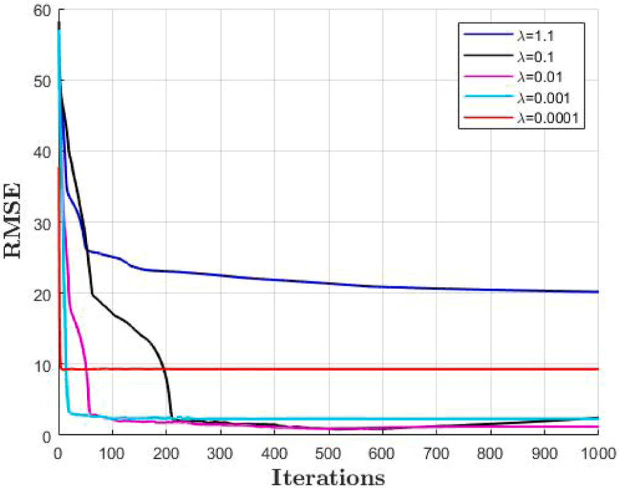
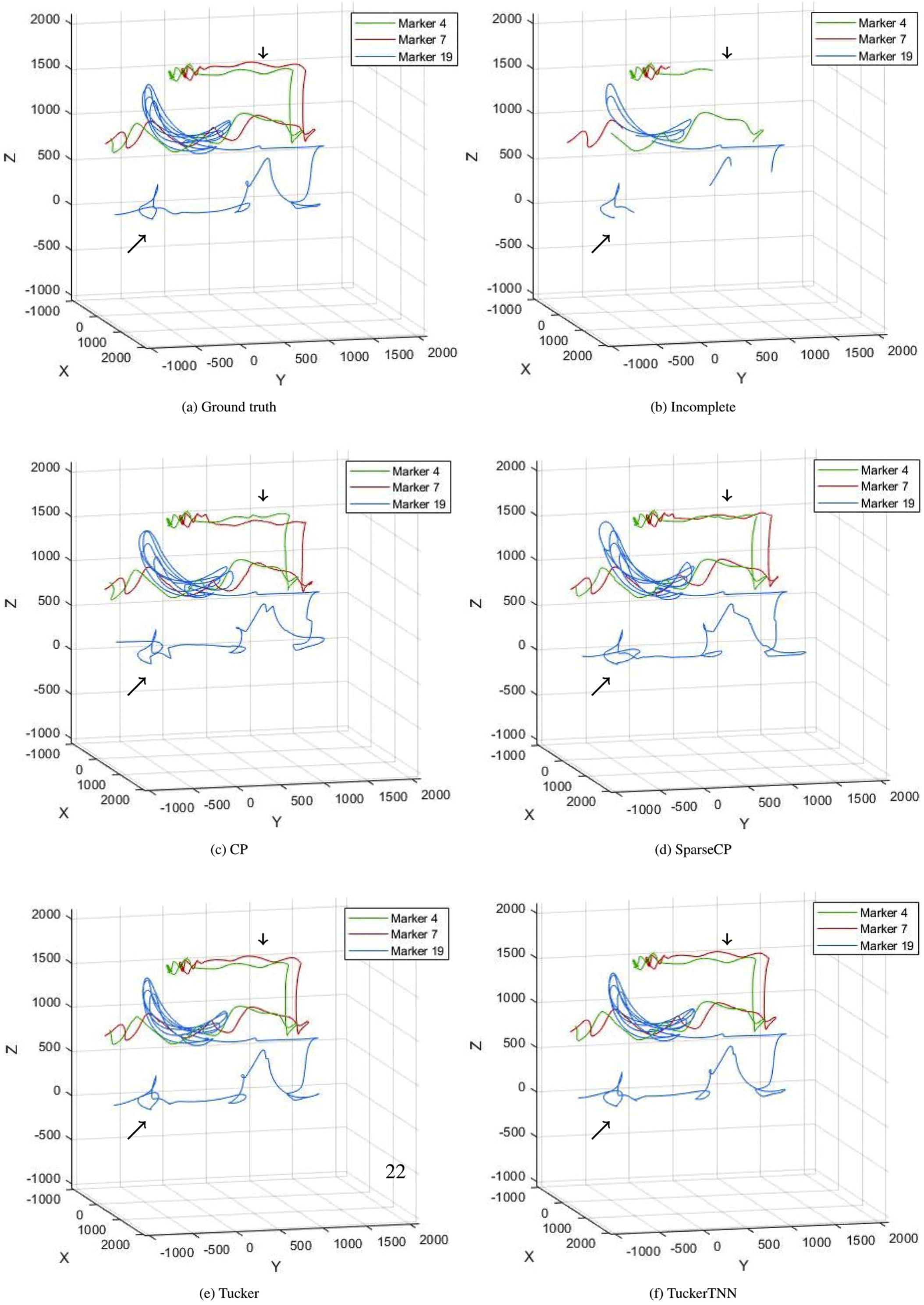


Fig. 5. Comparison of RMSE values during iterations across varying λ values.



(a) Ground truth

(b) Incomplete

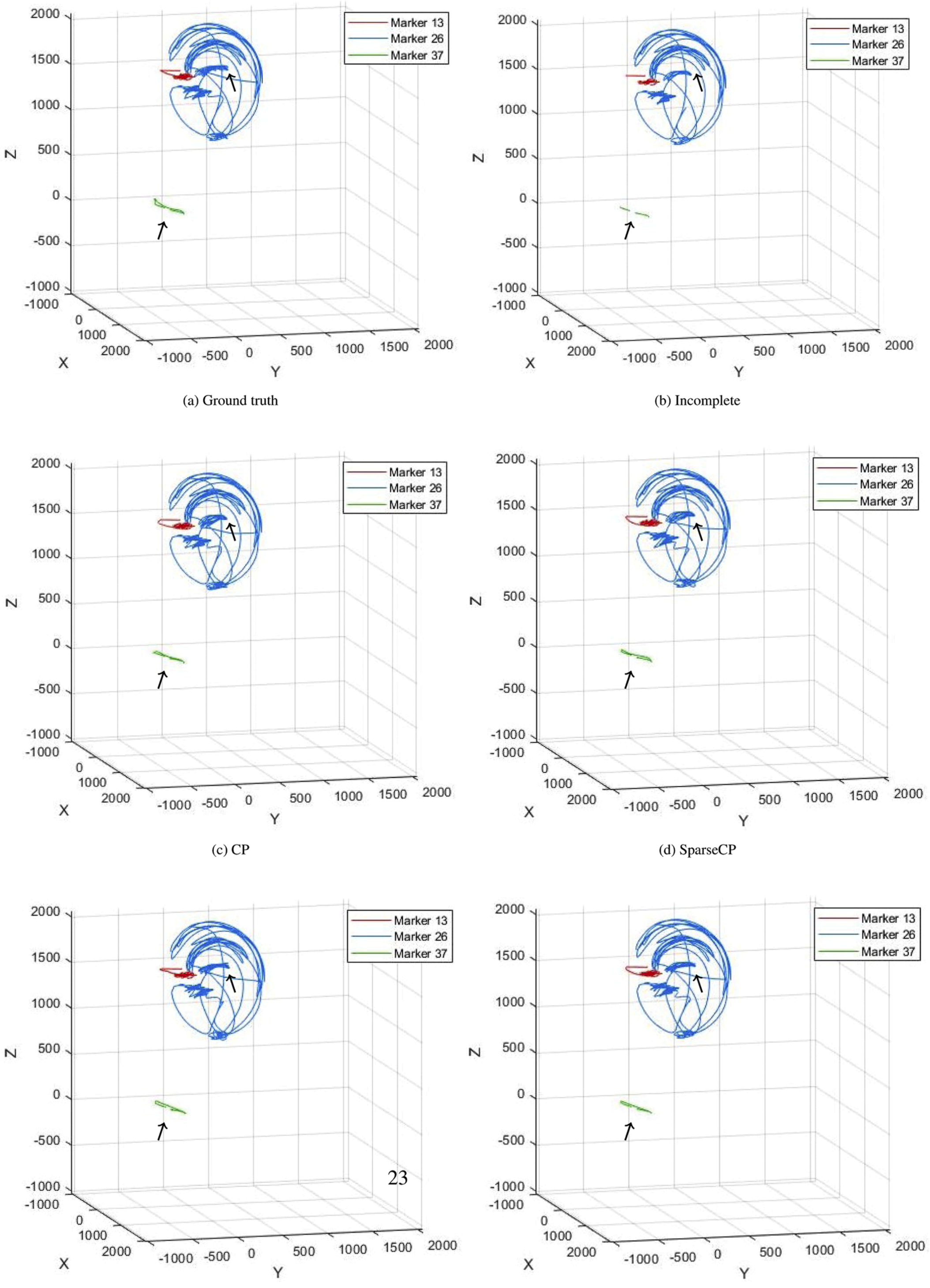
(c) CP

(d) SparseCP

(e) Tucker

(f) TuckerTNN

Fig. 6. Comparison results of recovered trajectories (3D positions over time) of HDM1 missing markers. (a) Ground truth trajectories, (b) Incomplete trajectories, (c) recovered trajectories by CP, (d) recovered trajectories by SparseCP, (e) recovered trajectories by Tucker, and (f) recovered trajectories by TuckerTNN.



(a) Ground truth

(b) Incomplete

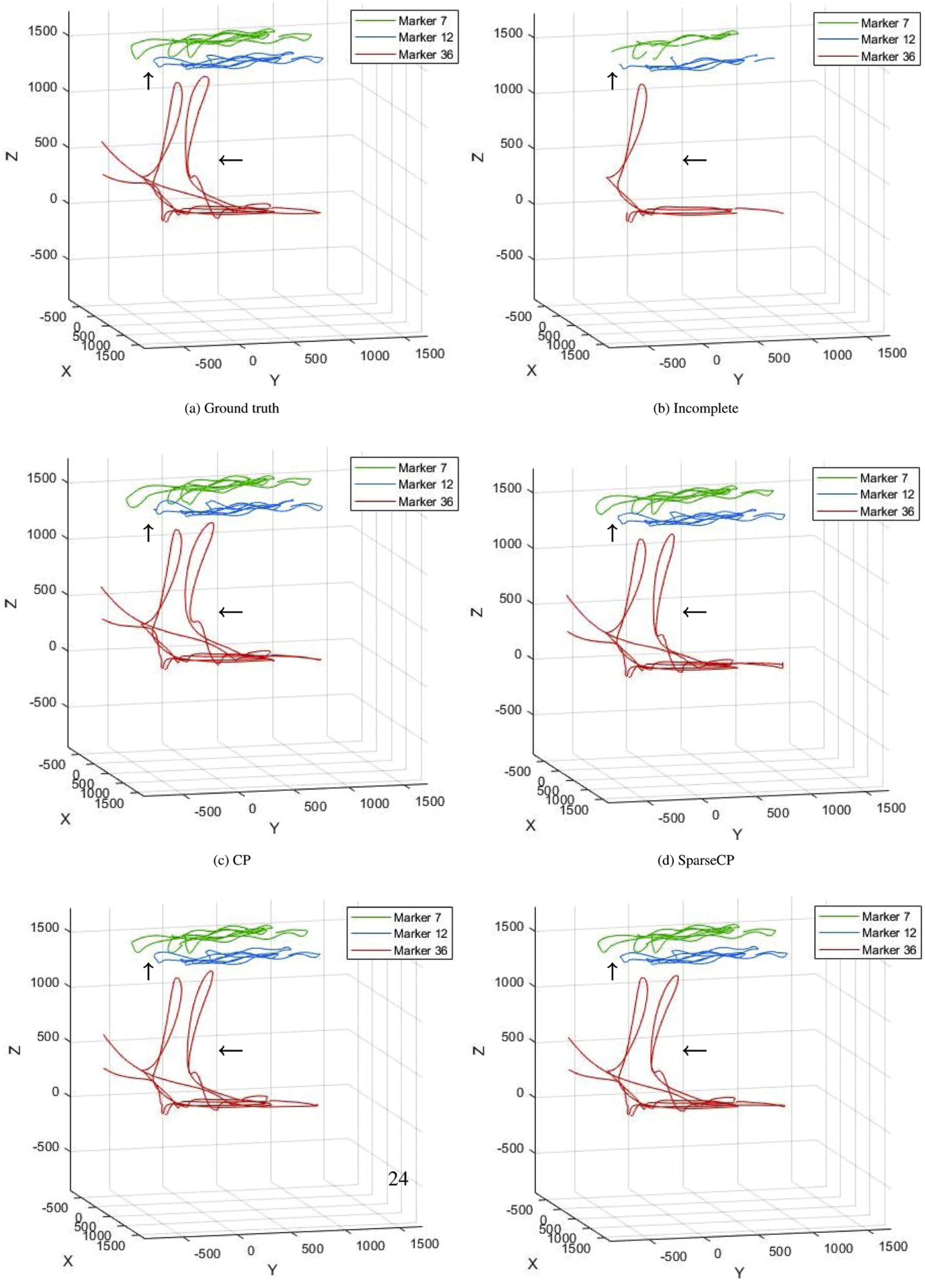
(c) CP

(d) SparseCP

(e) Tucker

(f) TuckerTNN

Fig. 7. Comparison results of recovered trajectories (3D positions over time) of HDM2 missing markers. (a) Ground truth trajectories, (b) Incomplete trajectories, (c) recovered trajectories by CP, (d) recovered trajectories by SparseCP, (e) recovered trajectories by Tucker, and (f) recovered trajectories by TuckerTNN.



(a) Ground truth

(b) Incomplete

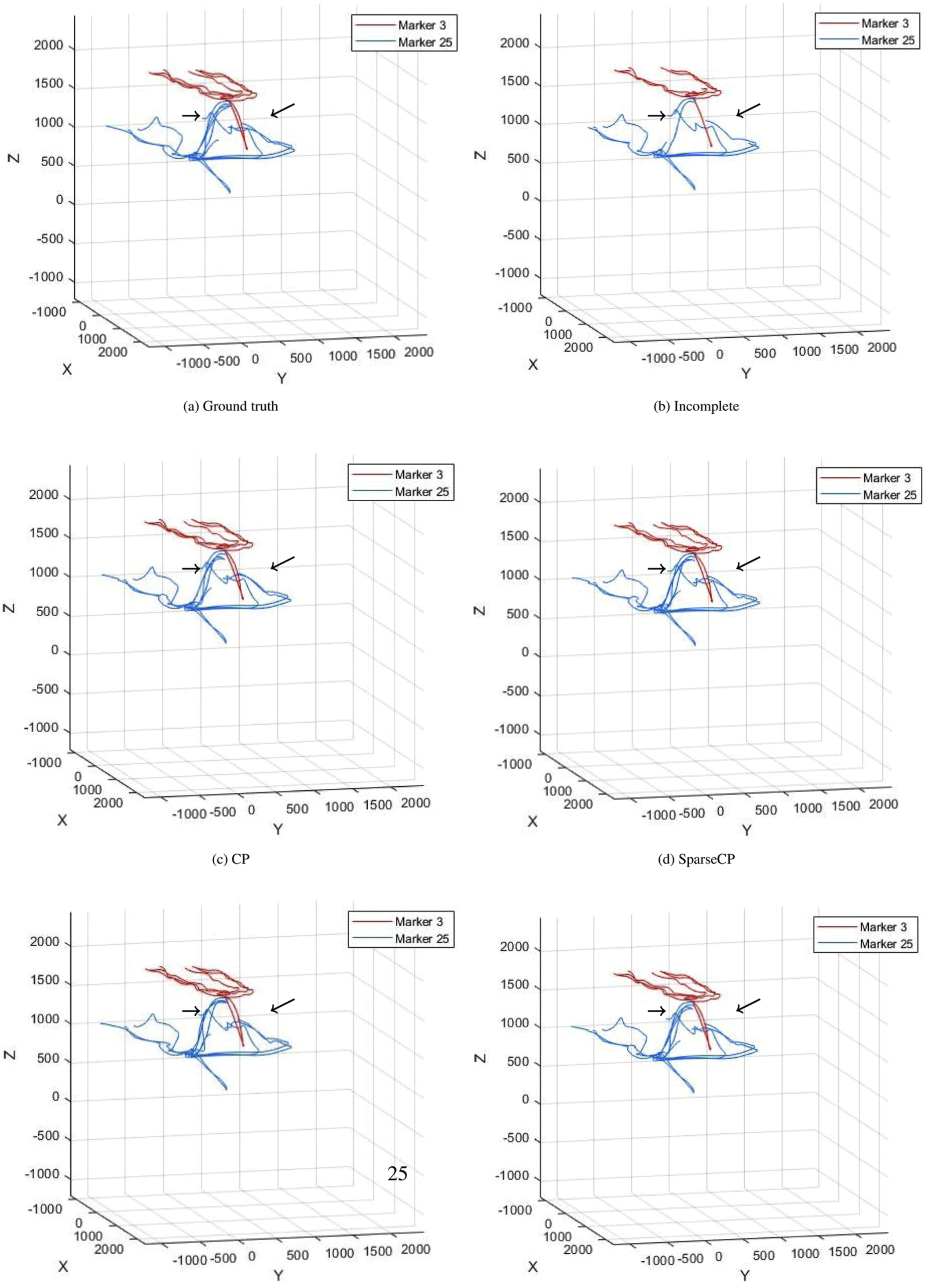
(c) CP

(d) SparseCP

(e) Tucker

(f) TuckerTNN

Fig. 8. Comparison results of recovered trajectories (3D positions over time) of HDM3 missing markers. (a) Ground truth trajectories, (b) Incomplete trajectories, (c) recovered trajectories by CP, (d) recovered trajectories by SparseCP, (e) recovered trajectories by Tucker, and (f) recovered trajectories by TuckerTNN.



(a) Ground truth

(b) Incomplete

(c) CP

(d) SparseCP

(e) Tucker

(f) TuckerTNN

Fig. 9. Comparison results of recovered trajectories (3D positions over time) of HDM4 missing markers. (a) Ground truth trajectories, (b) Incomplete trajectories, (c) recovered trajectories by CP, (d) recovered trajectories by SparseCP, (e) recovered trajectories by Tucker, and (f) recovered trajectories by TuckerTNN.

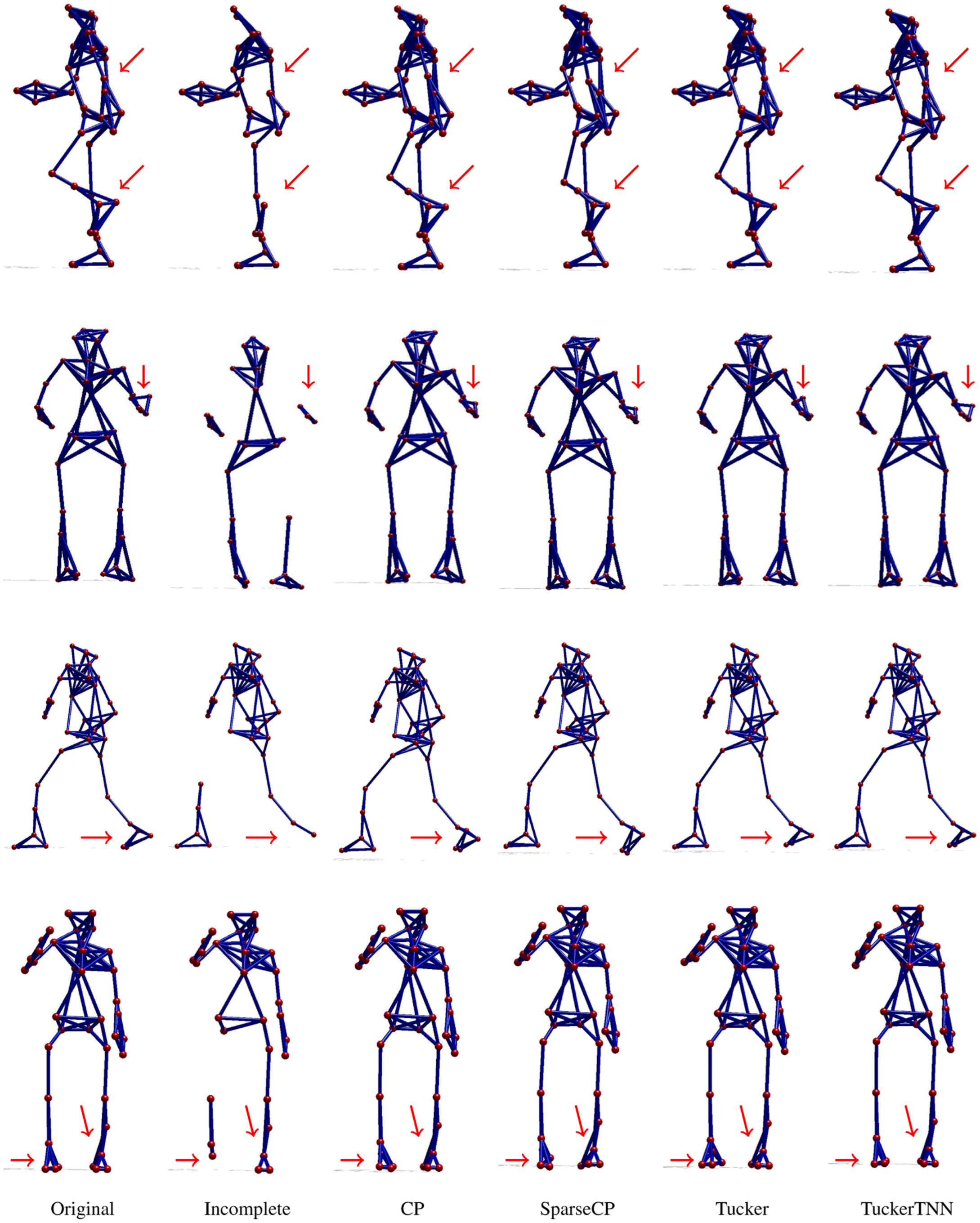


Fig. 10. 3D visualization comparing marker recovery results for the four sequences: HDM1, HDM2, HDM3, and HDM4. Red arrows indicate inaccuracies in the recovered markers.

Furthermore, we visually compare the markers recovery results of the four algorithms in Fig. 10 for the four motion sequences (HDM1, HDM2, HDM3, HDM4). We present the recovered results for a single frame from each motion sequence. In this experiment, 25 markers are missing, with the gap length set to 100, corresponding to approximately 20 gaps per marker. Red arrows highlight the markers that were recovered inaccurately. When comparing the four algorithms, CP and SparseCP show less accurate performance in recovering markers, as indicated by the arrows across all four datasets. TuckerTNN demonstrates more accurate marker recovery and outperforms Tucker, as highlighted, for example, in the last row of the figure.

Table 2
A comparison of RMSE, AvE, and STD for completing motion data (HDM1, HDM2, HDM3, and HDM4) conducted under different settings. Performance results from four algorithms-CP, SparseCP, Tucker, and TuckerTNN-are presented for each scenario, with the best outcomes highlighted in bold.

Data	NGaps	Methods	Length of Gaps = 100			Length of Gaps = 150			Length of Gaps = 200		
			RMSE	AvE	STD	RMSE	AvE	STD	RMSE	AvE	STD
HDM1	2	CP	2.558	2.11e-1	2.55	3.781	3.76e-1	3.78	4.405	5.24e-1	4.40
		SparseCP	2.349	2.06e-1	2.44	3.296	3.42e-1	3.29	4.786	5.19e-1	4.78
		Tucker	1.577	1.33e-1	1.57	1.835	2.02e-1	1.83	**1.759**	2.05e-1	**1.75**
		TuckerTNN	**9.023e-1**	**7.98e-2**	**0.90**	**1.480**	**1.46e-1**	**1.48**	1.825	**2.04e-1**	1.82
	5	CP	3.910	4.27e-1	3.91	5.517	7.15e-1	5.51	6.198	9.35e-1	6.19
		SparseCP	3.310	3.62e-1	3.30	4.664	6.2e-1	4.66	8.108	1.25	8.10
		Tucker	2.602	3.16e-1	2.60	2.456	3.46e-1	2.45	3.890	6.49 e-1	3.88
		TuckerTNN	**1.111**	**1.30e-2**	**1.11**	**1.651**	**2.23e-1**	**1.65**	**2.104**	**3.22e-1**	**2.10**
	10	CP	4.817	7.41e-1	4.81	6.977	1.25	6.97	8.338	1.76	8.33
		SparseCP	5.052	7.70e-1	5.05	7.315	1.39	7.31	7.737	1.59	7.73
		Tucker	2.688	3.90e-1	2.68	**2.571**	**4.45e-1**	**2.57**	3.634	8.16e-1	3.63
		TuckerTNN	**1.804**	**2.66e-1**	**1.80**	3.486	6.53e-1	3.48	**3.120**	**6.27e-1**	**3.12**
HDM2	2	CP	2.558	2.11e-1	2.55	3.781	3.76e-1	3.78	1.434	9.94e-2	1.43
		SparseCP	2.349	2.06e-1	2.44	3.296	3.42e-1	3.29	1.524	1.06e-1	1.52
		Tucker	7.728e-1	4.91e-2	0.77	8.66e-1	6.26e-2	0.86	6.969e-1	7.71e-2	0.69
		TuckerTNN	**3.754e-1**	**2.13e-2**	**0.37**	**5.582e-1**	**3.71e-2**	**0.55**	**6.841e-1**	**5.30e-2**	**0.68**
	5	CP	1.167	9.20e-2	1.16	1.197	7.33e-2	1.19	1.835	2.11e-1	1.83
		SparseCP	1.025	8.58e-2	1.02	1.239	7.42e-2	1.23	1.738	2.18e-1	1.73
		Tucker	6.178e-1	5.50 e-2	0.61	6.969e-1	7.711e-2	0.69	1.758	2.34e-1	1.75
		TuckerTNN	**5.275e-1**	**4.63e-2**	**0.52**	**6.807e-1**	**7.48e-2**	**0.68**	**9.032e-1**	**1.15e-1**	**0.90**
	10	CP	2.094	2.17e-1	2.09	6.977	1.25	6.97	2.784	4.53e-1	2.78
		SparseCP	1.977	2.01e-1	1.97	7.315	1.39	7.31	2.974	4.53e-1	2.97
		Tucker	1.042	1.30e-1	1.04	3.265	5.16e-1	3.26	2.388	4.26e-1	2.38
		TuckerTNN	**9.134e-1**	**1.03e-1**	**0.91**	**1.134**	**1.59e-1**	**1.13**	**1.457**	**2.36e-1**	**1.45**
HDM3	2	CP	1.111	7.02e-2	1.11	1.539	1.14e-1	1.53	1.806	1.48e-1	1.80
		SparseCP	1.179	7.18e-2	1.17	1.212	8.95e-2	1.21	1.872	1.55e-1	1.87
		Tucker	5.803e-1	3.63e-2	0.58	8.395e-1	6.59e-2	0.83	1.011	8.39e-2	1.01
		TuckerTNN	**4.439e-1**	**2.79e-2**	**0.44**	**5.802e-1**	**4.40e-2**	**0.58**	**6.926e-1**	**5.96e-2**	**0.69**
	5	CP	1.658	1.40e-1	1.65	2.332	2.59e-1	2.33	2.448	3.15e-1	2.44
		SparseCP	1.624	1.44e-1	1.62	2.276	2.36e-1	2.27	2.621	3.23e-1	2.62
		Tucker	2.364	2.01e-1	2.36	2.032	2.18e-1	2.03	2.671	3.32e-1	2.67
		TuckerTNN	**7.764e-1**	**6.82e-2**	**0.77**	**9.272e-1**	**1.00e-1**	**0.92**	**1.133**	**1.38e-1**	**1.13**
	10	CP	3.249	3.39e-1	3.24	3.920	4.75e-1	3.92	4.368	6.28e-1	4.36
		SparseCP	3.592	3.56e-1	3.59	3.864	4.67e-1	3.86	4.780	7.19e-1	4.78
		Tucker	2.426	2.68e-1	2.42	**1.823**	2.36e-1	**1.82**	3.578	5.47e-1	3.57
		TuckerTNN	**1.643e-1**	**1.69e-2**	**1.64**	2.012	**2.20e-1**	2.01	**2.476**	**2.89e-1**	**2.47**
HDM4	2	CP	1.032	5.70e-2	1.03	1.449	1.03e-1	1.44	1.501	1.20e-1	1.50
		SparseCP	1.066	5.97e-2	1.06	1.288	8.96e-2	1.28	1.622	1.28e-1	1.62
		Tucker	4.966e-1	2.85e-2	0.49	6.615e-1	4.59e-2	0.66	9.553e-1	7.61e-2	0.95
		TuckerTNN	**4.465e-1**	**2.51e-2**	**0.44**	**5.279e-1**	**3.65e-2**	**0.52**	**6.193e-1**	**4.95e-2**	**0.61**
	5	CP	1.537	1.28e-1	1.53	2.061	2.00e-1	2.06	2.325	2.68e-1	2.32
		SparseCP	1.611	1.33e-1	1.61	2.129	2.15e-1	2.12	2.528	2.76e-1	2.52
		Tucker	1.043	8.61e-2	1.04	1.240	1.22e-1	1.24	1.468	1.66e-1	1.46
		TuckerTNN	**6.366e-1**	**4.81e-2**	**0.63**	**8.632e-1**	**7.78e-2**	**0.86**	**1.135**	**1.22e-1**	**1.13**
	10	CP	2.346	2.61e-1	2.34	3.117	3.93e-1	3.11	3.750	5.61e-1	3.75
		SparseCP	2.228	2.38e-1	2.22	3.054	4.04e-1	3.05	3.864	5.61e-1	3.06
		Tucker	1.439	1.58e-1	1.43	2.428	2.99e-1	2.42	3.506	5.22e-1	3.50
		TuckerTNN	**9.056e-1**	**9.31e-2**	**0.90**	**1.363**	**1.61e-1**	**1.36**	**1.617**	**2.22e-1**	**1.61**

6.1. Convergence analysis

Based on the iterative behavior shown in Fig. 5, we analyze the convergence characteristics of the TuckerTNN completion method under different regularization parameters. All tested configurations demonstrate monotonic convergence behavior, with convergence characteristics that depend on the regularization parameter λ. Larger λ values ($\lambda = 1.1$) lead to faster convergence but result in higher RMSE, suggesting over-regularization. On the other hand, smaller λ values ($\lambda = 0.0001$) show slower convergence with better RMSE compared to ($\lambda = 1.1$), indicating under-regularization. The intermediate values (e.g., $\lambda = 0.01$) provide a balanced trade-off, achieving low RMSE with reasonable convergence speed, making them optimal for the TuckerTNN completion method.

Table 3
CPU time (in minutes) of the four methods for HDM1, HDM2, HDM3, and HDM4 sequences, respectively.

Motion data	Number of frames	CP	SparseCP	Tucker	TuckerTTN
HDM1	3990	4.739	7.636	**1.267**	*2.959*
HDM2	5920	6.583	10.750	**1.982**	*4.506*
HDM3	7671	11.425	15.320	**2.858**	*6.055*
HDM4	8500	12.610	16.403	**2.882**	*6.590*

6.2. Runtime performance analysis

Table 3 shows the CPU time (in minutes) of the four motion datasets. CP and SparseCP consistently require more computational time, with CP being the most demanding, especially for larger datasets like HDM3 and HDM4. Tucker and TuckerTNN offer more efficient performance, with Tucker being the fastest, particularly in HDM1. While TuckerTNN has slightly higher computational costs due to the added complexity of the temporal nuclear norm, it remains competitive compared to CP and SparseCP.

7. Conclusion

This paper proposes two gap-filling algorithms based on Tucker decomposition: the Tucker and TuckerTNN algorithms. We compare the performance of these algorithms with recent MoCap completion methods, namely CP and SparseCP. Our results demonstrate that the proposed Tucker and TuckerTNN algorithms consistently outperform CP and SparseCP in terms of accuracy and computational time. While Tucker provides the best computational efficiency, TuckerTNN offers improved accuracy due to the incorporation of temporal nuclear regularization, albeit with a slight increase in computational cost. These results underscore the effectiveness of the Tucker-based methods for gap recovery, particularly in complex and high-dimensional MoCap sequences, compared to CP-based algorithms. Tucker is the fastest algorithm, while TuckerTNN has slightly higher computational costs. However, it remains more accurate and faster than CP and SparseCP, while offering the best compromise between accuracy and efficiency.

Data availability

Data will be made available on request.

Acknowledgements

Competing Interests. The authors have no competing interests.

Funding. Carl Tryggers Stiftelse supports the first author's work under grant CTS 22:2196. The Swedish Research Council (VR) supports the second author's work under grant 2021–05393.

References

[1] S. Mohaoui, A. Dmytryshyn, Low-rank completion for motion capture data recovery: Approaches, constraints, and algorithms, Comput. Sci. Rev. 60 (2026) 100878.
[2] E. Martini, A. Calanca, N. Bombieri, Denoising and completion filters for human motion software: A survey with code, Comput. Sci. Rev. 58 (2025) 100780.
[3] H. Zhou, H. Hu, Human motion tracking for rehabilitation-a survey, Biomed. Signal Process Contr. 3 (1) (2008) 1–18.
[4] I. Takeda, A. Yamada, H. Onodera, Artificial intelligence-Assisted motion capture for medical applications: a comparative study between markerless and passive marker motion capture, Comput. Methods Biomech. Biomed. Engin. 24 (8) (2021) 864–873.
[5] E. Van der Kruk, M.M. Reijne, Accuracy of human motion capture systems for sport applications; state-of-the-art review, Eur. J. Sport. Sci. 18 (6) (2018) 806–819.
[6] G. Millour, A.T. Velásquez, F. Domingue, A literature overview of modern biomechanical-based technologies for bike-fitting professionals and coaches, Int. J. Sports Sci. Coaching 18 (1) (2023) 292–303.
[7] M. Field, D. Stirling, F. Naghdy, Z. Pan, Motion capture in robotics review, in: 2009 IEEE International Conference on Control and Automation, IEEE, 2009, pp. 1697–1702.
[8] R.Y.Q. Lai, P.C. Yuen, K.K.W. Lee, Motion capture data completion and denoising by singular value thresholding, in: Eurographics (Short Papers), 2011, pp. 45–48.
[9] C.-H. Tan, J. Hou, L.-P. Chau, Human motion capture data recovery using trajectory-based matrix completion, Electron Lett. 49 (12) (2013) 752–754.
[10] C.-H. Tan, J. Hou, L.-P. Chau, Motion capture data recovery using skeleton constrained singular value thresholding, Vis. Comput. 31 (2015) 1521–1532.
[11] B. Chen, H. Sun, G. Xia, L. Feng, B. Li, Human motion recovery utilizing truncated schatten p-norm and kinematic constraints, Inf Sci (Ny) 450 (2018) 89–108.
[12] Y. Feng, J. Xiao, Y. Zhuang, X. Yang, J.J. Zhang, R. Song, Exploiting temporal stability and low-rank structure for motion capture data refinement, Inf. Sci. 277 (2014) 777–793.
[13] W. Hu, Z. Wang, S. Liu, X. Yang, G. Yu, J.J. Zhang, Motion capture data completion via truncated nuclear norm regularization, IEEE Signal Process Lett. 25 (2) (2017) 258–262.
[14] J. Yang, X. Guo, W.M. Li, Kun, Y.-K. Lai, F. Wu, Spatio-temporal reconstruction for 3D motion recovery, IEEE Trans. Circuits Syst. Video Technol. 30 (6) (2019) 1583–1596.
[15] S.-J. Peng, G.-F. He, X. Liu, H.-Z. Wang, Hierarchical block-based incomplete human mocap data recovery using adaptive nonnegative matrix factorization, Comput. Graph. 49 (2015) 10–23.
[16] G. Xia, H. Sun, B. Chen, Q. Liu, L. Feng, G. Zhang, R. Hang, Nonlinear low-rank matrix completion for human motion recovery, IEEE Trans. Image Process. 27 (6) (2018) 3011–3024.
[17] Ø. Gløersen, P. Federolf, Predicting missing marker trajectories in human motion data using marker intercorrelations, PLoS ONE 11 (3) (2016) e0152616.

[18] Z. Li, H. Yu, H.D. Kieu, T.L. Vuong, J.J. Zhang, PCA-Based Robust motion data recovery, IEEE Access 8 (2020) 76980–76990.

[19] M.A.O. Vasilescu, Human motion signatures: analysis, synthesis, recognition, in: 2002 International Conference on Pattern Recognition, 3, IEEE, 2002, pp. 456–460.

[20] S. Mohaoui, A. Dmytryshyn, CP Decomposition-based algorithms for completion problem of motion capture data, Pattern Analysis and Applications 27 (4) (2024) 133.

[21] F.N. Fritsch, R.E. Carlson, Monotone piecewise cubic interpolation, SIAM J. Numer. Anal. 17 (2) (1980) 238–246.

[22] S.J. Howarth, J.P. Callaghan, Quantitative assessment of the accuracy for three interpolation techniques in kinematic analysis of human movement, Comput. Methods Biomech Biomed. Engin. 13 (6) (2010) 847–855.

[23] K. Dorfmüller-Ulhaas, Robust optical user motion tracking using a Kalman filter, Technical Report, Ins. für Informatik, Saarbrücken, Germany, 2003.

[24] L. Li, J. McCann, N. Pollard, C. Faloutsos, Bolero: a principled technique for including bone length constraints in motion capture occlusion filling, in: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2010, pp. 179–188.

[25] Q.M. Wu, P. Boulanger, Real-time estimation of missing markers for reconstruction of human motion, in: 2011 IEEE Virtual Reality Conference (VR), IEEE, 2011, pp. 161–168.

[26] W. Li, S. Peng, Y. Liu, W. Wang, Predicting missing marker trajectories in human motion data using marker intercorrelations and bayesian inference, PLoS ONE 11 (4) (2016) e0152616.

[27] W. Li, S. Peng, Y. Liu, W. Wang, A novel approach to solve the “missing marker problem” in marker-based motion capture, PLoS ONE 8 (11) (2013) e78689.

[28] X. Yang, A. Somasekharan, J.J. Zhang, Curve skeleton skinning for human and creature characters, Comput. Animat Virtual Worlds 17 (3–4) (2006) 281–292.

[29] W. Cheng, R. Cheng, L. Xiaoyong, D. Shuling, Automatic skeleton generation and character skinning, in: 2011 IEEE International Symposium on VR Innovation, IEEE, 2011, pp. 299–304.

[30] J.D. Carroll, J.-J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of “Eckart-Young” decomposition, Psychometrika 35 (3) (1970) 283–319.

[31] L.R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (3) (1966) 279–311.

[32] I.V. Oseledets, Tensor-train decomposition, SIAM Journal on Scientific Computing 33 (5) (2011) 2295–2317.

[33] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.

[34] J. Levin, Three-mode factor analysis, Psychol. Bull. 64 (6) (1965) 442.

[35] L. De Lathauwer, B. De Moor, J. Vandewalle, A multilinear singular value decomposition, SIAM J. Matrix Anal. Appl. 21 (4) (2000) 1253–1278.

[36] P.M. Kroonenberg, J. De Leeuw, Principal component analysis of three-mode data by means of alternating least squares algorithms, Psychometrika 45 (1) (1980) 69–97.

[37] Y. Xu, W. Yin, A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion, SIAM J Imaging Sci 6 (3) (2013) 1758–1789.

[38] L. Grippo, M. Sciandrone, On the convergence of the block nonlinear Gauss–Seidel method under convex constraints, Oper. Res. Lett. 26 (3) (2000) 127–136.

[39] H. Attouch, J. Bolte, B.F. Svaiter, Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–Seidel methods, Math. Program 137 (1) (2013) 91–129.

[40] Y.E. Nesterov, A method for solving the convex programming problem with convergence rate O (1/kˆ 2), in: Dokl. Akad. Nauk Sssr, 269, 1983, pp. 543–547.

[41] A. Beck, M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM J. Imaging Sci. 2 (1) (2009) 183–202.

[42] J.-F. Cai, E.J. Candès, Z. Shen, A singular value thresholding algorithm for matrix completion, SIAM J. Optim. 20 (4) (2010) 1956–1982.

[43] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, A. Weber, Documentation Mocap Database HDM05, Technical Report CG-2007-2, Universität Bonn, 2007.