Multimedia Systems (2020) 26:157–172
https://doi.org/10.1007/s00530-019-00633-9

REGULAR PAPER

RMoCap: an R language package for processing and kinematic analyzing motion capture data

Tomasz Hachaj[1] · Marek R. Ogiela[2]

Received: 18 August 2018 / Accepted: 19 March 2019 / Published online: 31 August 2019
© The Author(s) 2019

Abstract
Package RMoCap is an advanced open-source tool for scientists, engineers and computer graphics familiar with R language who work with motion capture (MoCap) technology. Package provides them with MoCap data handling, statistical processing, visualizing and analysis. Package uses well-established MoCap file exchange format and can be easily integrated with most of the motion analysis workflows. Among functions available in RMoCap, there are procedures for conversion between hierarchical and direct kinematic models, data averaging, correcting direction of motion, 3D interactive visualization and advanced analysis using Dynamic Time Warping. This paper covers all advanced algorithms that are implemented in RMoCap. This article also introduces direct to hierarchical kinematic conversion algorithm that is a new, never-before-published method. We also introduce extension of motion direction correction method that makes possible to process data that does not contain information about acceleration. All examples showed in this paper can be reproduced using replication code and data sources that are included to this package.

Keywords Motion capture · Kinematic model · Hierarchical kinematic · BVH · Motion processing · Motion analysis · Motion averaging · R language

1 Introduction

Motion capture (MoCap) technology generates motion description that is composed of a set of time varying signals that describe positions of body joints. This modern technology has many important applications; among them are: sport data analysis, medicine, biomechanics and computer graphic. The *RMoCap* package is devoted to statistical processing and kinematic analyzing of this type of data.

The structure of our paper goes as follows: the rest of the first section is devoted to presentation of the state of the art in MoCap kinematic analysis. We will also initially show how RMoCap covers various MoCap applications and we will introduce its novel functionalities. Section 1.3 presents description of algorithms notation we use in this paper. Sections 1.4 and 1.5 are technical details about input data format and package installation notes. Sections 2, 3, 4, 5 and 6 are devoted to the most important algorithms of RMoCap. Sections 2 and 3 covers mapping from hierarchical to direct kinematic model and vice versa. Section 4 is about motion direction correction that is applied to improve results of body displacement estimation. Section 5 describes motion averaging algorithm which uses several recordings of same activity to generate the single motion pattern. The sixth section covers motion analysis procedure that enables comparison of two MoCap recordings to detect, measure and visualize important kinematic differences between them. The last, seventh contains summary and discussion.

1.1 State of the art in kinematic analysis

Kinematics is the branch of mechanics that deals with the motion of the bodies and system without considering the force [28]. Forward kinematics is specified by the joints parameters and kinematic equations that are used to compute

Communicated by M. Katsurai.

✉ Tomasz Hachaj
tomekhachaj@o2.pl

Marek R. Ogiela
mogiela@agh.edu.pl

[1] Institute of Computer Science, Pedagogical University of Cracow, Kraków, Poland

[2] Cryptography and Cognitive Informatics Research Group, AGH University of Science and Technology, Kraków, Poland

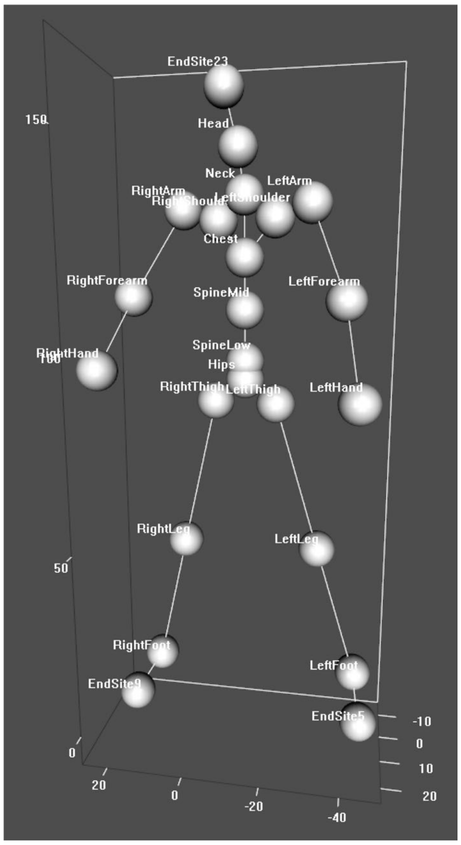

Fig. 1 A three-dimensional interactive plot that visualizes single frame of 3D motion capture data

the position of the end effector from specified value for each joint parameter (in this paper, we do not take into account an inverse kinematic problem which is not a case in MoCap technology). Let us define the Direct Kinematic (DK) model/description as motion notation in which body joints positions are directly described by 3D coordinates of each body joint (no further calculation is required to determine they spatial positions). This model does not take into account rotation, so it treats each body joint as a material point in space rather than a rigid body. In Hierarchical Kinematic (HK) model/description, we define the motion as a set of 3D rotations of body joints that are organized in hierarchy (so called kinematic chain)—a tree structure that has a root joint. Distances (offsets) between body joint and its parent are defined relatively to the parent joint. Also rotations are defined relatively to the parent joint. The root joint, that does not have the parent joint, beside rotation also contains information about translation of the whole body. An example visualization of joints organized in hierarchy can be seen in Fig. 1. In that figure, a root joint is a Hips joint and it has three children: SpineLow, RightThigh and LeftThigh; RightThigh has a child RightLeg, etc. To get spatial coordinates of body joints from HK model, we have to recalculate it to DK. DK model is often more useful for kinematic data analysis. However, for example, computer graphic often utilizes HK, because it has the information about joints orientation (rotation). Cheap MoCap hardware like Kinect generates data in DK model. To get joints orientation, we need to recalculate DK to HK model. To do so, we need to have a prior definition of joints hierarchy that is compatible with DK model (that has same number of joints).

The *RMoCap* package is dedicated to work on MoCap data that are either in DK or HK model. Our package can freely convert HK into DK and DK into HK and is operational on various types of public datasets. The DK into HK conversion algorithm is a new, never-before-published authors' method.

Nearly, each motion capture hardware manufacturer supplies users with a basic graphical interface and/or application programming interface that enables conversion of a raw data into popular data formats that can be processed by third-party software. Also, there are often dedicated packages for computing basic kinematic parameters like time of motion, trajectory length, linear velocity, acceleration, angles between body joints, etc. Those parameters are often used by scientists to generate tabularized values that later become reference values for various motion activities [16, 17, 33]. From the statistical and computer science perspective, those are very basic operations. Those statistic can be generated by several common programming language instructions (for example *summary* command in *R* language). Due to this, our package does not implement operations that are available as generic *R* language functions and we will not discuss those approaches later on.

Among the most interesting and fruitful methods of motion analysis are motion capture averaging methods and motion path matching approaches. The motion averaging is a process in which as an input data we take several recordings of a person who performs same action. After applying averaging, we generate a single recording that should maximize value of similarity measure between averaged recording and each input recording (so motion averaging is an optimization procedure). The aim of motion averaging is to remove small, random noises that might be present in MoCap recording. Motion averaging does not remove systematic errors that might visualize some common motion inaccuracies. Depending of application of MoCap analysis, an inaccuracy might be an effect of some disabilities caused by illness (medical and rehabilitation application) or wrong performance of technique (sport application). Our package implements method presented in paper [12] that enables averaging 3D MoCap data using dynamic time warping barycenter averaging (DBA) [24]. It operates in quaternion space using Markley averaging approach instead of barycenter [19]. Package *RMoCap* is the first open-source release of this method.

The motion path matching approaches analyze trajectory of certain body joints and align one MoCap data to another to find some important differences between them. There are many papers that describe application of that type of analysis in 1D and 2D [7, 21, 22, 29–31]. In our package we implement 3D trajectory comparison method based on Dynamic Time Warping (DTW) approach that is able to find and visualize the highest local differences in kinematic chain.

The IMU-based (Inertial Measurement Unit) MoCap hardware, that typically contains accelerometers, gyroscopes and magnetometers, is capable to measure the acceleration of the objects that are attached to costume. By numerical integration, we can estimate the distance that those sensors moved starting from the known, initial position. Due to limited precision of IMU (both static and dynamic), some unavoidable drifts from the ”real” body position are continuously introduced. However, the most important problem with IMU-based MoCap is the fact, that those three earlier mentioned measuring sensors (accelerometer, gyroscope and magnetometer) are not enough to measure the body displacement is 3D space relatively to the ground (we will call it later a body displacement). The only reliable displacement returned by those sensors is relative motion to root joint of HK model. To measure body displacement, some additional sensors have to be introduced, for example, vision-based (cameras) or pressure sensors located on feet. Vision-based system is not reliable as long as it is not a professional MoCap with at least the same price as IMU system itself. Pressure sensors operate under some heuristic that often introduces errors in estimation body displacement vector. Knowing that the *RMoCap* package introduces function that is a heuristic that can estimate the body displacement of MoCap data. The proposed heuristic is a novel extension of the method proposed in [10]. The extension in *RMoCap* package makes method capable to process data that do not contain information about acceleration.

1.2 Open-source software packages dedicated to motion capture analyzing

In this subsection, we will only discuss package for MoCap data analysis. There are dozens of open packages for converting 2D image data into 3D MoCap (for example [5]) and for MoCap-based pattern recognition [9]; however, they are designed to solve different problems than our algorithms. Also kinetic (not kinematic) analysis performed, for example, by MotionLab is out of our scope [25].

Our package is capable of freely converting not only HK into DK model (there are many examples of that in number of programming languages) but also DK into HK which is a more complex optimization task. For our best knowledge, this is a first open-source publicly available package that implements this type of conversion.

R language has very little packages directly dedicated for MoCap data analysis. As far as we know, there is only *mocap* [27] package that enables loading ASF/AMC files and converting them into a list data structure. The visualization functions of that package seem to be based on old version of third-party libraries and are not operational any more. The second package we found, *mocapGrip* [14] encapsulates a *python* motion capture project dedicated for annotations analysis, which is utilized mostly in linguistic and psychology. Both of those packages do not contain functions that can be helpful in motion processing and kinematic analysis. DBA algorithm is implemented in *R* language package [26] and DTW in package [8]; however, both of those implementations do not allow to use quaternion data. Also, DTW implementation is not flexible enough to access all algorithm parameters that we need. Due to this, *RMoCap* has its own implementations of both algorithms, independent from both packages.

Among important open-source packages created in other than R programming languages is Mokka,[1] which is a motion capture kinematic and kinetic analyzer with graphical interface that implements basic plots, media integration and data importing functions. It utilizes functions from an open source framework Biomechanical ToolKit (BTK) [2]. BTK supports C++, Matlab and Python programming languages. OpenSim is a software to create and analyze dynamic simulations of movement [4]. An open source toolbox MoBILAB [23] can be used for analysis and visualization of mobile brain/body imaging data. HuMAns toolbox [32] includes a biomechanical model of a complete human body and proposes a set of versatile tools for modeling, capture, analysis and simulation of human and humanoid motion (it is an open-source software, distributed under the GPL License). MoCap Toolbox [3] is a set of functions written in Matlab for analyzing and visualizing motion capture data. It covers basic visualization and analysis approaches, such as general data handling, creating stick-figure images and animations, kinematic analysis (mean and standard deviation of velocity and acceleration), and performing Principal Component Analysis (PCA) on movement data.

Although all mentioned software contain useful methods, they do not cover new functionalities that are included in *RMoCap* that we initially presented in Sect. 1.1. All of them will be discussed in detail in the rest of this paper.

1.3 Algorithms notation

All algorithms that are implemented in *RMoCap* package are described using standard mathematical notation or, if this notation might be too clumsy, with code in *R* language or

[1] http://biomechanical-toolkit.github.io/mokka/.

pseudocode similar to *R* language. In case of mathematical notation, matrices are named with capital letter, numbers are in lowercase, angles are lowercase Greek letters and vectors are lowercase letters with arrow. In *R*/R pseudocode, all fields of complex data types (i.e., named columns of lists, matrices, data frames) are accessed with $ operator. Vectors are defined with parentheses, while accessing object with certain index in data frame, list or matrix require square brackets. Dot symbol "." might be a part of the name of the variable. Hash symbol "#" begins a single-line commentary. We use notation "/* */" for commentaries in pseudocode.

1.4 Test datasets and accepted input file formats

RMoCap package contains several MoCap recordings that can be used to test algorithms we have implemented. The MoCap data were recorded using Shadow 2.0 wireless motion capture system, which is high-end professional IMU-based solution. We use this hardware for our everyday work. More about recording process can be learned from [12]. Motion recordings are example karate techniques performed by a world and national class professional (medalists) karate athletes. There are both short recordings of single karate kicks and also longer karate kata motion sequences. Of course, our algorithms can read and operate on other recordings. There are two basic file formats *RMoCap* uses: Biovision Hierarchy file format (BVH), that is a very popular format to hold HK model [20] and also comma-separated vector (CSV) file format. Requirements for CSV are proper naming of columns:

- CSV must have a column with name Time that shows the acquisition time of single frame (in milliseconds);
- rotation or/and translation data columns for each body joint, each column of this type has to have name starting with body joint name (for example RightLeg) and ending .Dx,.Dy,.Dz for translation data (for example RightLeg. Dx, the unit should be centimeters). Rotation data columns are .Rx, .Ry, .Rz (for example RightLeg.Rx, the unit should be Euler angles in degrees – not radians).

In case of *R* language, CSV files are often stored in memory in *data.frame* structure. If *data.frame* columns have same number of joints definition in DK model as HK in BVH file, data can be easily converted from DK into HK (see example in Sect. 3). Most of the current MoCap systems and motion repositories store data in at least one of those formats. It requires basic knowledge of *R* to prepare a *data.frame* imported from CSV to work with *RMoCap*.

1.5 Installing R package RMoCap

The *R* package *RMoCap* [11] is distributed under the GPL-3 license. It depends on the R packages smoother v. 1.1 [13], rgl v. 0.99.16 [1], RSpincalc v. 1.0.2 [6], subplex v. 1.5.4 [15], signal v. 0.7.6 [18], compiler v. 3.5.0.

The R package *RMoCap* is hosted by GitHuba[2] and its dependencies are available at https://CRAN.R-project.org/ and can be installed as follows:

```
#if you have not installed "devtools" package
R> install.packages("devtools")
R> devtools::install_github("browarsoftware/RMoCap")
```

2 Mapping from HK to DK model

In this section, we will present how to recalculate MoCap data from HK to DK model. In KH model, the rotation of each joint is governed by three-dimensional rotation described by Euler angles. Let us assume that the order of rotation is Z, Y, X and rotation angles are α, β, γ, respectively. The rotation matrix has following form:

$$M_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \quad M_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}, \quad M_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

and the final rotation:

$$R_{ot} = M_z \cdot M_y \cdot M_x. \tag{2}$$

Root joint is the only joint that in HK that has both rotation $T^{\text{Root}}_{\text{rans}}$ and translation D^{Root}_{isp} data. Taking into account offset of the root joint $O^{\text{Root}}_{\text{ffset}}$, the final displacement (3D position in DK model) of root joint D^{Root}_{xyz} is:

$$D^{\text{Root}}_{xyz} = D^{\text{Root}}_{isp} + O^{\text{Root}}_{\text{ffset}}. \tag{3}$$

Then we need to calculate the translation matrix of root joint $T^{\text{Root}}_{\text{rans}}$ that will be used later:

$$T^{\text{Root}}_{\text{rans}}(4 \times 4) = \begin{bmatrix} \begin{bmatrix} R^{\text{Root}}_{ot}(3 \times 3) \end{bmatrix} & \begin{bmatrix} D_{xyz}(3 \times 1) \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}, \tag{4}$$

[2] https://github.com/browarsoftware/RMoCap.

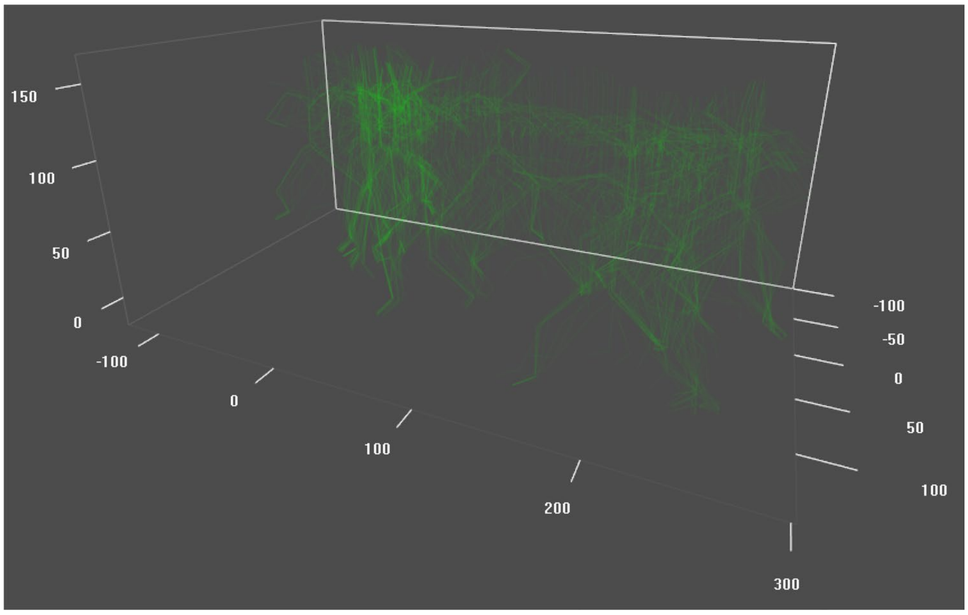

Fig. 2 A three-dimensional interactive plot that visualize 3D motion capture data

(3×3) and (3×1) indicates the dimensionality of matrices.

The calculation of final displacements (positions in DK model) of all joints has to be made according to hierarchy indicated in HK. This means that at first, we need to calculate translation matrix of root joint, then positions of its direct descendants, then translation of descendants of those descendants, etc. until we calculate translation matrices of end site joints (joints without children).

A joint with index i which is neither root joint nor end site has only a rotation data. To calculate its translation matrix, we need to multiply translation matrix of the parent joint of joint i $T_{\text{rans}}^{\text{Parent}(i)}$ by a matrix composed of rotation matrix and offset of joint i:

$$T_{\text{rans}}^{\text{Joint}_i}(4 \times 4) = T_{\text{rans}}^{\text{Parent}(i)} \cdot \begin{bmatrix} \begin{bmatrix} R_{ot}^{\text{Joint}_i}(3 \times 3) \end{bmatrix} \begin{bmatrix} O_{\text{ffset}}^{\text{Joint}_i}(3 \times 1) \end{bmatrix} \\ 0 \quad 0 \quad 0 \quad 1 \end{bmatrix}. \tag{5}$$

End site joints do not have rotation data; due to this translation matrix of end site joint j is calculated by multiplying translation matrix of parent joint $T_{\text{rans}}^{\text{Parent}(j)}$ by matrix composed of offset of end site j:

$$T_{\text{rans}}^{EndSite_j}(4 \times 4) = T_{\text{rans}}^{\text{Parent}(j)} \cdot \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} O_{\text{ffset}}^{EndSite_j}(3 \times 1) \end{bmatrix} \\ 0 \quad 0 \quad 0 \quad 1 \end{bmatrix}. \tag{6}$$

The displacement D_{xyz} of all types of joints is in first three rows of fourth column of their T_{rans} matrix:

$$D_{xyz} = [T_{\text{rans}}^{1,4}, T_{\text{rans}}^{2,4}, T_{\text{rans}}^{3,4}]. \tag{7}$$

Above procedures are implemented in *RMoCap* package in function *read.mocap*. This function loads data stored in BVH format from disc and generates object of class mocap that contains all original HK data and data frame with DK model.

```
R> library(RMoCap)
R> data("heian.nidan.bvh")
R> f <- file("heian.nidan.bvh")
R> writeChar(con = f, object = heian.nidan.bvh)
R> close(f)
#read hierarchical model stored in BVH file
R> heian.nidan <- read.mocap("heian.nidan.bvh")
#plot kinematic data
R> plot(heian.nidan, frame = 1, my.color = "white",
  alpha = 1, spheres = TRUE, print.text = TRUE)
R> plot(heian.nidan)
```

Visualization of 3D data can be seen on Figs. 1 and 2.

3 Mapping from DK to HK model

Let us face the following problem: we have 3D coordinates of body joints (direct kinematic description) and a hierarchical kinematic model that is compatible with direct one (it has the same number of joints). We want to recalculate direct kinematic parameters to hierarchical so that they describe the same motion.

Mapping from DK to HK is, similarly as mapping from HK to DK, an iterative procedure, that starts from the root node and explores its children setting appropriate rotation coefficients. In HK, each joint holds information about its rotation in Euler angles. Additionally, root joint (the joint that does not have a parent joint) has a translation data.

In algorithm proposed in this paper we assume that the rotation order of Euler angles will be *ZYX* and there is only one root joint with at least two direct descendants.

The below calculation has to be repeated for each frame of motion capture recording.

Among the basic steps of this mapping algorithm is to find 3D rotation matrix between vectors. As it is not a trivial procedure, we will describe this method in the following section.

3.1 Finding n-dimensional rotation matrix between vectors

The method of finding n-dimensional rotation matrix R_n that rotates vector $\vec{x}$ to $\vec{y}$ is not a basic but already known algebraic procedure. Let us assume that $\vec{x}$ and $\vec{y}$ are linearly independent. If this condition is false, the rotation is represented by identity matrix.

At first we will generate 2D rotation matrix R_2 such that: $x \cdot R_2$ is linearly dependent to y.

$$R_2 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \tag{8}$$

where α is an angle on the plane between $\vec{x}$ to $\vec{y}$, $\cos(\alpha) = \frac{\vec{x} \circ \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$, $\sin(\alpha) = \sqrt{1 - \cos^2(\alpha)}$ and $\circ$ is a dot product.

```
Pseudocode 1: Translation calculation
  Input: HK – a MoCap data structure in hierarchical kinematic model
  DK – a MoCap data structure in direct kinematic model.
  Output: Translated HK
1 HK$Root$RawDxyz ← DK$Root$Dxyz - HK$Root$Offset
```

In the next step, we will generate orthonormal pairs of vectors $\vec{a}, \vec{b}$ from $\vec{x}, \vec{y}$ using Gram–Schmidt process.

We need to normalize vector $\vec{x}$:

$$\vec{a} = \frac{\vec{x}}{|\vec{x}|}, \tag{9}$$

and then by projection of vector $\vec{y}$ onto the line spanned by $\vec{a}$

$$\vec{b} = \vec{y} - (\vec{a} \circ \vec{y}) \cdot \vec{a}. \tag{10}$$

The projection matrix on subspace designated by unit vectors $\vec{a}, \vec{b}$ is:

$$P = \vec{a} \cdot \vec{a}^{\mathrm{T}} + \vec{b} \cdot \vec{b}^{\mathrm{T}}. \tag{11}$$

The projection matrix onto complemented subspace n-2 subspace is:

$$P' = I_n - \vec{a} \cdot \vec{a}^{\mathrm{T}} - \vec{b} \cdot \vec{b}^{\mathrm{T}}, \tag{12}$$

where I_n is n – dimensional identity matrix.

Because the rotation happens in subspace designated by vectors $\vec{a}, \vec{y}$ we can get n-dimensional substitute of R_2 by changing the base of R_2 to be n-dimensional and by completing it with P':

$$R_n = I_n - \vec{a} \cdot \vec{a}^t - \vec{b} \cdot \vec{b}^t + [\vec{a}\vec{b}] \cdot R_2 \cdot [\vec{a}\vec{b}]^{\mathrm{T}}. \tag{13}$$

Knowing how to calculate R_n, we can go to the first step of DK to HK mapping that is determination of rotation of root joint.

3.2 Root joint rotation calculation in HK model

Among all joints in HK model, the root joint is a special case because of two factors:

- This is the only joint that has a translation data,
- as this joint does not have a parent we need two vectors in 3D space to unambiguously define its rotation.

Let HK be a MoCap data structure in hierarchical model, while DK a MoCap data structure in direct kinematic model. Calculation of translation data is straightforward (see below pseudocode):

Finding rotation of the root joint is, however, more complex. At first we need to find any two direct descendants of the root. If there are more than two, the rest is irrelevant for this algorithm.

We choose one of these descendants (Child1) and calculate rotation matrix that rotates vector designated by offset of HK root joint onto vector designated as difference between Child1 and Root joints coordinates in DK model.

```
Pseudocode 2: Root rotation calculation – part 1
   Input: HK – a MoCap data structure in hierarchical kinematic model
   DK – a MoCap data structure in direct kinematic model.
   Output: Rotation matrix of HK root
 1 parent.dxyz ← DK$Root$Dxyz
 2 child1.dxyz ← DK$Child1$Dxyz
 3 map ← vector.to.unit(child1.dxyz - parent.dxyz)
 4 to.map ← vector.to.unit(HK$Root$Offset)
   /* algorithm from Section 3.1                              */
 5 Rx2y ← rotation.matrix(to.map, map)
   /* map == to.map %*% Rx2y                                  */
```

Where vector.to.unit is a function that normalizes the vector and rotation.matrix is a function that implements an algorithm from Sect. 3.1.

Having rotation matrix *Rx2y*, we can now calculate quaternion after rotation by which we will rotate one vector onto another. In *R*, we can do it with function DCM2EA from package *RSpincalc*. Pseudocode 2 finds matrix that maps vector defined by an offset of root joint onto vector designated by position of child joint. The rotation matrix Rx2y as it was said in Sect. 3.1 operates on the 2D plane and there is no guarantee that the initial solution we found also maps all other direct descendants of the root joint. That is because the solution we initially obtained might be rotated around $\overrightarrow{map}$. To find the appropriate 3D rotation that also maps other children of root we have to find additional rotation that applied after the first one correctly aligns both child-parent vectors. To do this, we can apply the following algorithm:

```
Pseudocode 3: Root rotation calculation – part 2
    Input: HK – a MoCap data structure in hierarchical kinematic model
    DK – a MoCap data structure in direct kinematic model
    Rx2y – rotation matrix from Pseudocode 2.
    Output: Rotated HK
    /* matrix - to - quaternion conversion                     */
 1  q1 ← DCM2EA(Rx2y, 'zyx')
    /* apply q1 to rotate whole KH                             */
 2  HK.help1 ← UpdateModel(HK, q1)
    /* also update DK coordinates                              */
 3  axis ← HK$Child1$Dxyz - HK$Root$Dxyz
    /* Find unknown.angle                                      */
 4  Function function.to.minimize(unknown.angle)
 5  begin
        /* axis angle to quaternion                            */
 6      q2 ← EV2Q(axis, unknown.angle)
 7      q3 ← q2 %Q*% q1
        /* applay q1 for KH rotation                           */
 8      HK.help2 ← UpdateModel(HK.help1, q3)
 9      return (euc.dist(DK$Child2$Dxyz,
10      HK.help2$Child2$Dxyz)
11  end
12  optimal.angle ← Simplex(function.to.minimize(unknown.angle = 0))
        /* Calculate return quaternion                         */
13  return.q ← EV2Q(axis, optimal.angle) %Q*% q1
    /* update HK model                                         */
14  HK ← UpdateModel(HK, return.q)
```

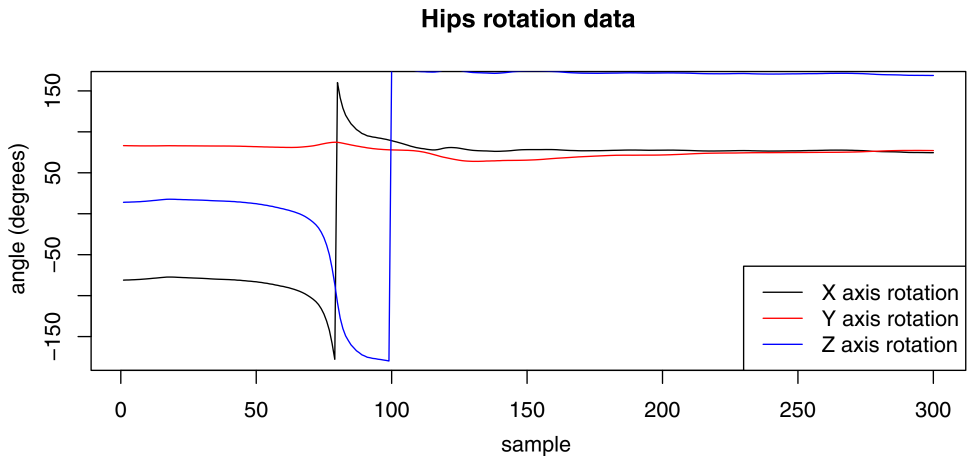

Fig. 3 This plot presents example rotation angles of hip joints in HK model generated from input DK data frame

Optimization from Pseudocode 3 can be done with Simplex method that is implemented for example in function *subplex* from package *subplex*. Functions EV2Q that calculates quaternion from axis angle is from package *RSpincalc*. The variable HK in last line of above pseudocode contains correctly mapped root joint.

3.3 All other joints

After processing the root joint, we iterate through all its children. Those children become parents of the following algorithm. All joints that have parent joint (joints with children and end sites) are processed in the same way. Because we want to find rotation angles relatively to the parent joint, we need to know the transformation matrix Trans of the parent joint. Trans is calculated in the same way as it is in mapping procedure from HK to DK model (see Sect. 2). See Pseudocode 4 for more details.

Pseudocode 4: Children rotation calculation

```
  Input: HK – a MoCap data structure in hierarchical kinematic model
  DK – a MoCap data structure in direct kinematic model.
  Output: Rotation matrix of a child
1 parent.n.dxyz $\leftarrow$ DK$Parent.n$Dxyz
2 child.m.dxyz $\leftarrow$ DK$Child.m$Dxyz
3 map $\leftarrow$ vector.to.unit(child.m.dxyz - parent.n.dxyz)
  /* inverse of transform matrix of parent of parent joint,
     Trans matrix can be found using equations from Section 2
     */
4 map $\leftarrow$ invert(HP$Parent.n$Parent$Trans) %*% map
5 to.map $\leftarrow$ vector.to.unit(HK$child.m$Offset)
  /* algorithm from Section 3.1                              */
6 Rx2y.n $\leftarrow$ rotation.matrix(to.map, map)
  /* map == to.map %*% Rx2y                                  */
```

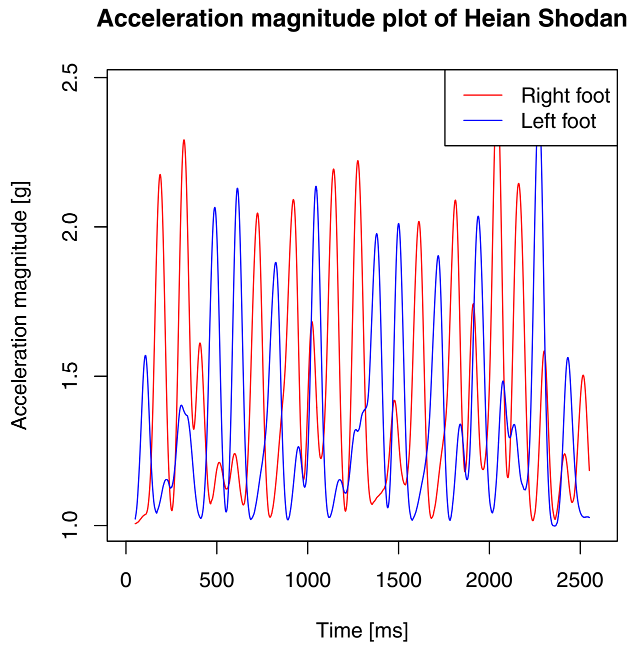

Fig. 4 Acceleration magnitude plot obtained for MoCap data from Heian Shodan motion sample

As the result in variable *Rx2y.n*, we have the rotation matrix of analyzed joint. After processing a parent joint (Parent.n), we find all children of this joint. Those joints are processed with Pseudocode 4. The program ends when there are no more joints in HK model.

Algorithms described in this section are implemented in *RMoCap* package in function df.to.bvh.

The following *R* code generates HK from DK and creates plot presented in Fig. 3. As can be seen Euler angles may contains nonlinearities (near 76 and 100 sample) that are present due to periodic domain of trigonometric functions; however, the effector trajectory remains smooth.

```
R> library(RMoCap)
R> data("header.mocap")
R> data("heian.yondan")
R> input.skeleton <- header.mocap
R> df.to.save <- heian.yondan[1:300,]
R> first.frame <- df.to.bvh(input.skeleton,
        df.to.save, debug.messages = TRUE)
R> plot(first.frame$skeleton$Joints[[1]]$Rxyz[,1],
        type = "l", col = "black",
        xlab = "sample", ylab = "angle (degrees)")
R> lines(first.frame$skeleton$Joints[[1]]$Rxyz[,2],
        type = "l", col = "red")
R> lines(first.frame$skeleton$Joints[[1]]$Rxyz[,3],
        type = "l", col = "blue")
R> legend("bottomright",
legend=c("X axis rotation",
        "Y axis rotation", "Z axis rotation"),
        col=c("black", "red", "blue"), lty = 1)
R> title("Hips rotation data")
```

The more advanced examples are presented in Supplements—R replication code.

4 Motion direction correction

In Sect. 1.1, we have described the important issue of IMU-based MoCap technology: it does not measure body displacement directly—it has to be estimated. To overcome this limitation, *RMoCap* package implements extension of heuristic approach proposed in [10]. After applying this method, we can estimate body displacement (motion) relatively to environment.

The proposed heuristic works in one of the two possible versions: first version that uses acceleration data and the second, in which we directly indicate which body joint remains immobile during whole motion.

4.1 Correction with acceleration data

This approach has three main assumptions:

1. During the whole motion always one feet remains on the same ground level (a person is not jumping, climbing etc.);
2. A person does not perform body displacement by sliding on the ground;
3. A base on which person is moving is flat.

At first we calculate magnitudes of acceleration of right M_R and left foot M_L and we smooth them with Gaussian kernel. M_R and M_L are vectors that are obtained as a result of smoothing right foot $\overrightarrow{RightFoot.a}$ and left foot $\overrightarrow{LeftFoot.a}$ acceleration signals, respectively.

$$\begin{aligned} M_R &\leftarrow \left|\overrightarrow{RightFoot.a}\right| \bigotimes G_\sigma, \\ M_L &\leftarrow \left|\overrightarrow{LeftFoot.a}\right| \bigotimes G_\sigma, \end{aligned} \tag{14}$$

where $\overrightarrow{RightFoot.a}$ is an acceleration signal (a vector of 3D acceleration vectors), G_σ is a Gaussian kernel and $\bigotimes$ is convolution operator.

In Fig. 4, we present acceleration magnitude plot obtained for MoCap data from Heian Shodan motion sample from *RMoCap* package. It can be clearly observed which foot is one with higher magnitude of acceleration (this foot moves) moves and which one with lower magnitude of acceleration remains on the ground.

In the next step for each sample i of M_R and M_R, we check if $M_R[i] > M_L[i] \wedge M_R[i+1] > M_L[i+1]$. If it is true, there is a motion of right foot in this sample and left foot remains on the ground. We perform displacement correction of all $\overrightarrow{ALL.Dxyz}$ displacement vectors (of all body joints vectors) in our dataset:

$$\begin{aligned} &\overrightarrow{ALL.Dxyz}[i : samples.count(\overrightarrow{LeftFoot.Dxyz})] \leftarrow \\ &\quad \overrightarrow{ALL.Dxyz}[i : samples.count(\overrightarrow{LeftFoot.Dxyz})] \\ &\quad - (\overrightarrow{LeftFoot.Dxyz}[i+1] - \overrightarrow{LeftFoot.Dxyz}[i]), \end{aligned} \tag{15}$$

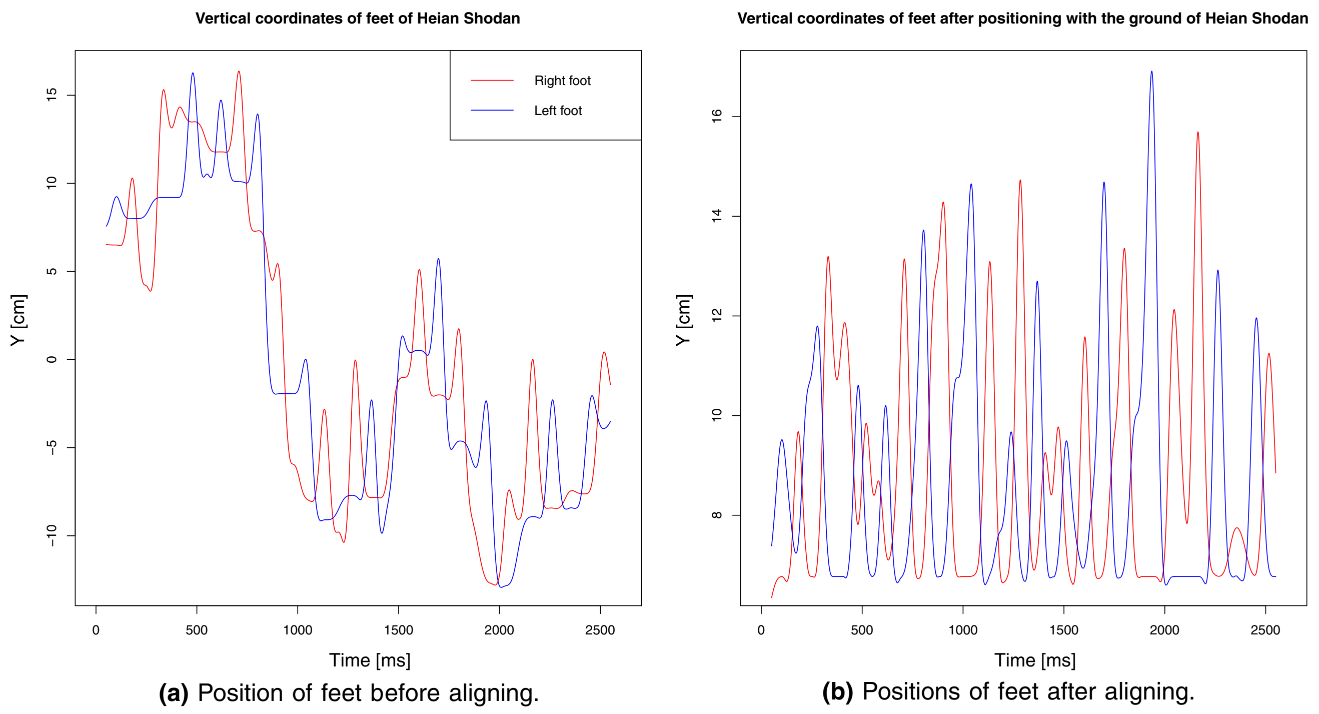

Fig. 5 Aligning feet positioning with ground level

where $i : samples.count(\overrightarrow{LeftFoot.Dxyz})$ is a range of samples to be corrected starting from ith sample and ending on the last index of sample from the MoCap dataset. $\overrightarrow{LeftFoot.Dxyz}[i]$ is an ith sample vector of left foot displacement.

The second condition we take into account is $M_R[i] < M_L[i] \wedge M_R[i+1] < M_L[i+1]$. If it is true, there is a motion of left foot in this sample and right foot remains on the ground. We perform displacement correction of all $\overrightarrow{ALL.Dxyz}$ displacement vectors (of all body joints vectors) in our dataset:

$$\begin{aligned}&\overrightarrow{ALL.Dxyz}[i : samples.count(\overrightarrow{RightFoot.Dxyz})] \leftarrow \\ &\quad \overrightarrow{ALL.Dxyz}[i : samples.count(\overrightarrow{RightFoot.Dxyz})] \\ &\quad - (\overrightarrow{RightFoot.Dxyz}[i+1] - \overrightarrow{RightFoot.Dxyz}[i]).\end{aligned} \tag{16}$$

In the next step, we need to correct the vertical (Y) coordinate of tracked person so that foot that remains on the ground would be on the same ground level $GroundPosition.Dy$, which might not be a true condition in original dataset (see Fig. 5a). Let us assume, that initial position of feet is correct (both feet are on the ground level). For each motion sample i of M_R and M_R we check if $M_R[i] \geqslant M_L[i]$. If it is true, there is a motion of right foot in this sample and left foot remains still on the ground. We perform displacement correction of all $ALL.Dy$ displacement vector coordinates (of all body joints vectors) in our dataset:

$$\begin{aligned}&ALL.Dy[i] \leftarrow ALL.Dy[i] \\ &\quad - (LeftFoot.Dy[i] - GroundPosition.y).\end{aligned} \tag{17}$$

If $M_R[i] < M_L[i]$ the processing is analogical to above.

The overall results for large motion sample are visualized in Fig. 6. Red silhouettes are original while green are corrected by our approach. An example R code that utilizes *calculate.kinematic* and plots results analogical to Figs. 4 and 5 is presented below.

```
R> library(RMoCap)
R> data("header.mocap")
R> data("heian.shodan")
R> heian.shodan.corrected <-
       calculate.kinematic(heian.shodan,
       show.plot = "TRUE", plot.title = "Heian Shodan")
```

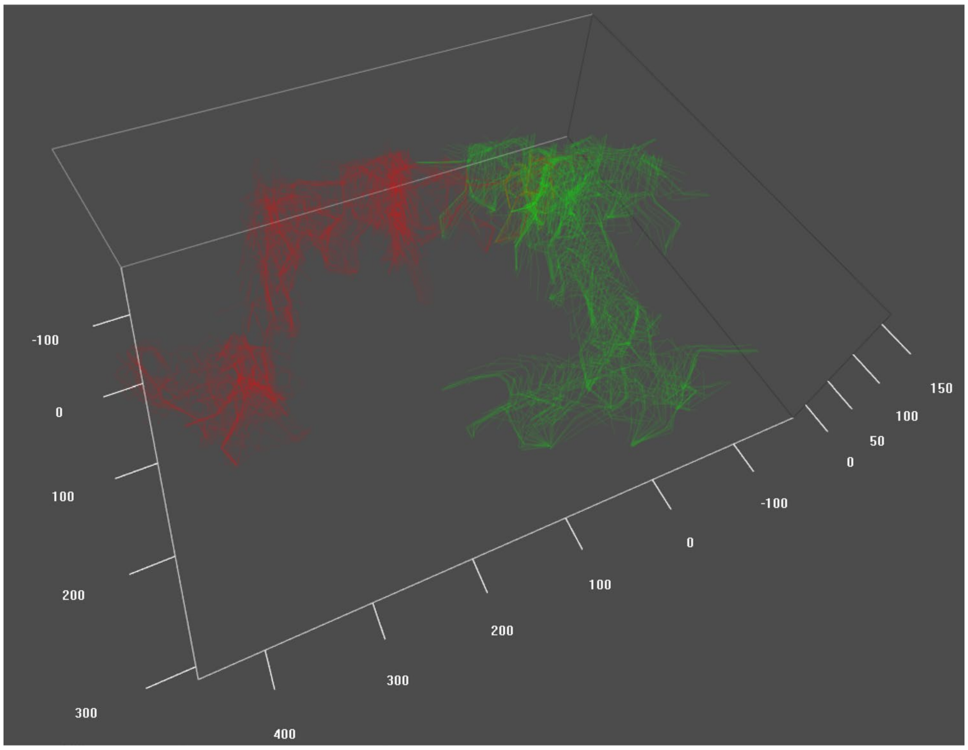

Fig. 6 This figure presents original motion (red silhouettes) and corrected motion by calculate.kinematic function (green silhouettes) (color figure online)

4.2 Correction without acceleration data

In this approach of motion direction correction, we indicate which body joint should be immobile (to have constant position) during the whole motion. We might want to do it in two main scenarios: we do not have acceleration data or/and we want to analyze motion relatively to some body joint. After indicating the constant joint we apply substantially the same approach as in (15); however, instead of $\overrightarrow{LeftFoot.Dxyz}$, we use displacement vector of the chosen joint. The application of this method will be discussed in section about Motion analysis in *RMoCap* package.

5 Motion averaging

The role of motion averaging algorithm is to generate single MoCap data from many motion recordings of the same activity performed by the same person to generate a single motion pattern. This motion pattern should be free of all random errors that might happen between repetitions of the same move. The averaging algorithm implemented in *RMoCap* is *mocap.averaging* method proposed in paper [12]. Because this method is already fully described in that open access paper, here we will only presents its basis and additional coverage checking method, which was not present in earlier published paper.

A *mocap.averaging* utilizes dynamic time warping barycenter averaging (DBA) heuristic algorithm [24] that is applied to data represented in HK model. Algorithm does not take into account translation of root joint which is not averaged. Each iteration of DBA utilizes dynamic time warping (DTW), where cost function is defined as:

$$d_{\text{istance}}(q_i, q_j) = 1 - \left|q_i \circ q_j\right|, \tag{18}$$

where q_i, q_j are normalized quaternions, ∘ is a dot product and | | is an absolute value (q and -q correspond to the same rotation).

DBA is applied separately to each three-dimensional rotation signal and rotation of each joint is recalculated to quaternions. Because this MoCap averaging algorithm uses quaternions the barycenter averaging (that happens during each DBA iteration) is replaced by norm-preserving quaternion averaging Markley's approach [19]. Markley's approach works as follow. Let us assume that we have to average k quaternions. At first, we iteratively generate matrix in the following form:

$$M = \begin{cases} M_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \\ M_i = M_{j-1} + w_j \cdot q_j \cdot q_j^{\mathrm{T}} \end{cases} \tag{19}$$

where q_j is – ith quaternion to be averaged, $j \in \{1, ..., k\}$ and w_j–ith weight of weight vector. To calculate the average

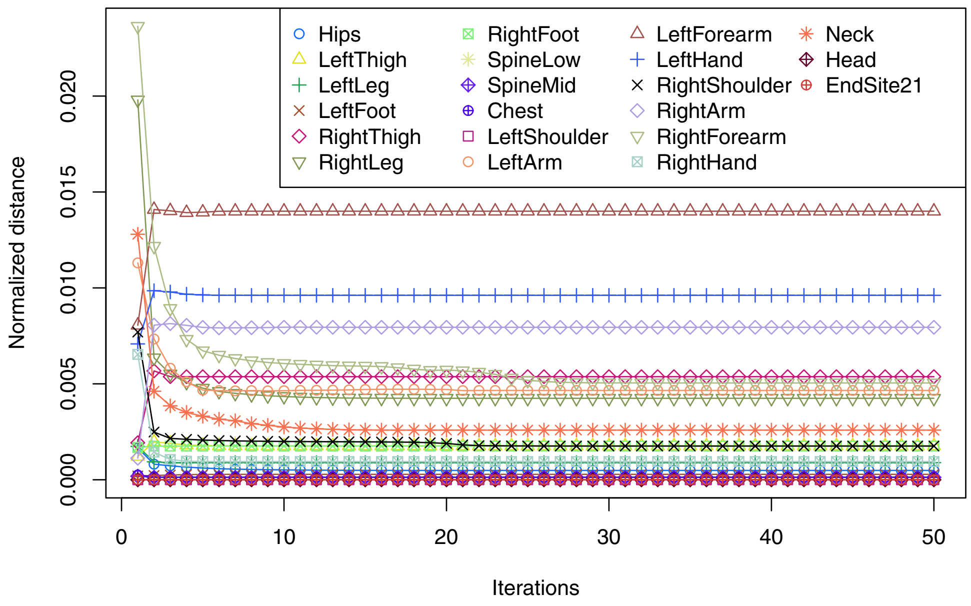

Fig. 7 This figure presents normalized distance after each iteration of DBA. Those distances are stored in DBA distance matrix for each body joint that is processed during averaging

quaternion q_{avg}, we need to find eigenvector of M corresponding to the maximum eigenvalue.

RMoCap has implementation of both weighted and not weighted (where $\forall j \in \{1, \dots, k\} w_j = 1$) Markley's algorithms in functions *wavg.quaternion.markley* and *avg.quaternion.markley*, respectively.

The stop criterion of averaging algorithm for each rotation signal is either maximal iterations count or coverage of the algorithm. The coverage happens when a module of difference of normalized DTW distances in DBA cost matrix of actual and previous iteration is below certain epsilon value or when normalized DTW distances in DBA cost matrix of actual iteration is below one tenth of epsilon value. The return signal from DBA has number of samples equals to the longest input signal from the averaged group. To smooth the final solution, we apply to each return signal weighted Markley's algorithm in the process similar to convolution filtration with Gaussian kernel (see [12] for more details).

An example R code that utilizes *mocap.averaging* and draws averaging plots for each body joint is presented below. The plot analogical to the one we will obtain in R is presented in Fig. 7.

```
R> library(RMoCap)
R> data("mawashi.geri.right.list")
R> myList <- list()
#Use only data frames
R> for (a in 1:length(mawashi.geri.right.list))
R> {
R> myList[[a]] <-mawashi.geri.right.list[[a]]$data.frame
R> }
#set seed to have repeatable results
R> set.seed(123)
R> res.data <- mocap.averaging(myList, 50, eps = 0.000001)
R> plot(res.data)
#save results in BVH file
 skel <- set.data.frame(mawashi.geri.right.list[[1]],
          res.data$fullData)
 write.bvh(path = "avg.mawashi.bvh",
          skeleton.helper = skel)
```

6 Motion analysis

The motion analysis that is performed in *RMoCap* package is based on motion paths alignment and comparison. The aim of this comparison is to find those fragments in motion paths, where there are locally maximal differences between input and reference data. The analysis is performed on chain of body joints that are defined before analysis (they might be the same as in HK model, however this is not obligatory). We need to indicate which joint is the joint in the end of the chain (we will call it end joint) and which are its parent joints. We may want to align either 3D coordinates of reference and input body joints (for example a foot) or angles on the plain between vectors designated by tracked body joints (for example knee flexion angle) or Euler angles in certain coordinate frame (for example rotation angle towards X axis between thigh and leg), however the end joint has to be analyze as 3D coordinates. Euclidean distance is commonly used to all of those cases. This type of data evaluation is based on evaluation of so called DTW align function (DTWaf) which is defined as follow:

- It is a vector or real values;
- It has the length of the DTW warping path (warping paths *path*1 and *path*2 that maps reference data on input data and vice versa has the same length);
- The value in each sample is calculated using following formula (see below pseudocode):

DTWaf holds information about distances between aligned signals. *distance*(x,y) is a distance function that was used in DTW. If more than one sample of one signal was warped onto another sample, DTWaf takes maximal distance value.

The motion analysis algorithm works as follow:

1. Both MoCaps have to be initially translated and rotated so that persons on them face same direction. To perform rotation, we choose vectors $v_{\text{reference}}$ and v_{input} in reference and input data derived from coordinates of body joints (for example, *LeftThigh* – *RightThigh*) and we align those two vectors by rotation around Y axis. The optimal angle of rotation minimizes Euclidean distance between $v_{\text{reference}}$ and v_{input}. The optimization is done using Simplex method. This procedure works if the translation of the root joint in MoCap data is zero.
2. Because root joint translation in original data is zero, we need to perform motion direction correction without using acceleration data of both reference and input data. To do so, we use the procedure described in Sect. 4.2.
3. Next, we translate input data so that it initially has same coordinates of selected joint $\overrightarrow{Joint.Dxyz_{\text{input}}}$ as the same joint from reference data $\overrightarrow{Joint.Dxyz_{\text{reference}}}$. After this operation both input and reference body motions start from the same point in space.

$$\begin{aligned}\overrightarrow{ALL.Dxyz_{\text{input}}}[i] &\leftarrow \overrightarrow{ALL.Dxyz_{\text{input}}}[i]\\ &- (\overrightarrow{Joint.Dxyz_{\text{input}}}[1] - \overrightarrow{Joint.Dxyz_{\text{reference}}}[1]).\end{aligned} \tag{20}$$

Pseudocode 5: DTW align function calculation

```
Input: path1, path2 – DTW warping paths
reference.data – reference data
input.data – input data
Output: DTW align function (DTWaf)
1  id ← 1
2  id.prev ← -1
3  for a in 1:length(path1) do
4      id ← path2[a]
5      x ← reference.data[path1[a]]
6      y ← input.data[path2[a]]
7      if id != id.prev then
8          DTWaf[id] ← distance(x, y)
9      end
10     else
11         DTWaf[id] ← max(DTWaf[id.prev], distance(x, y))
12     end
13     id.prev ← id
14 end
```

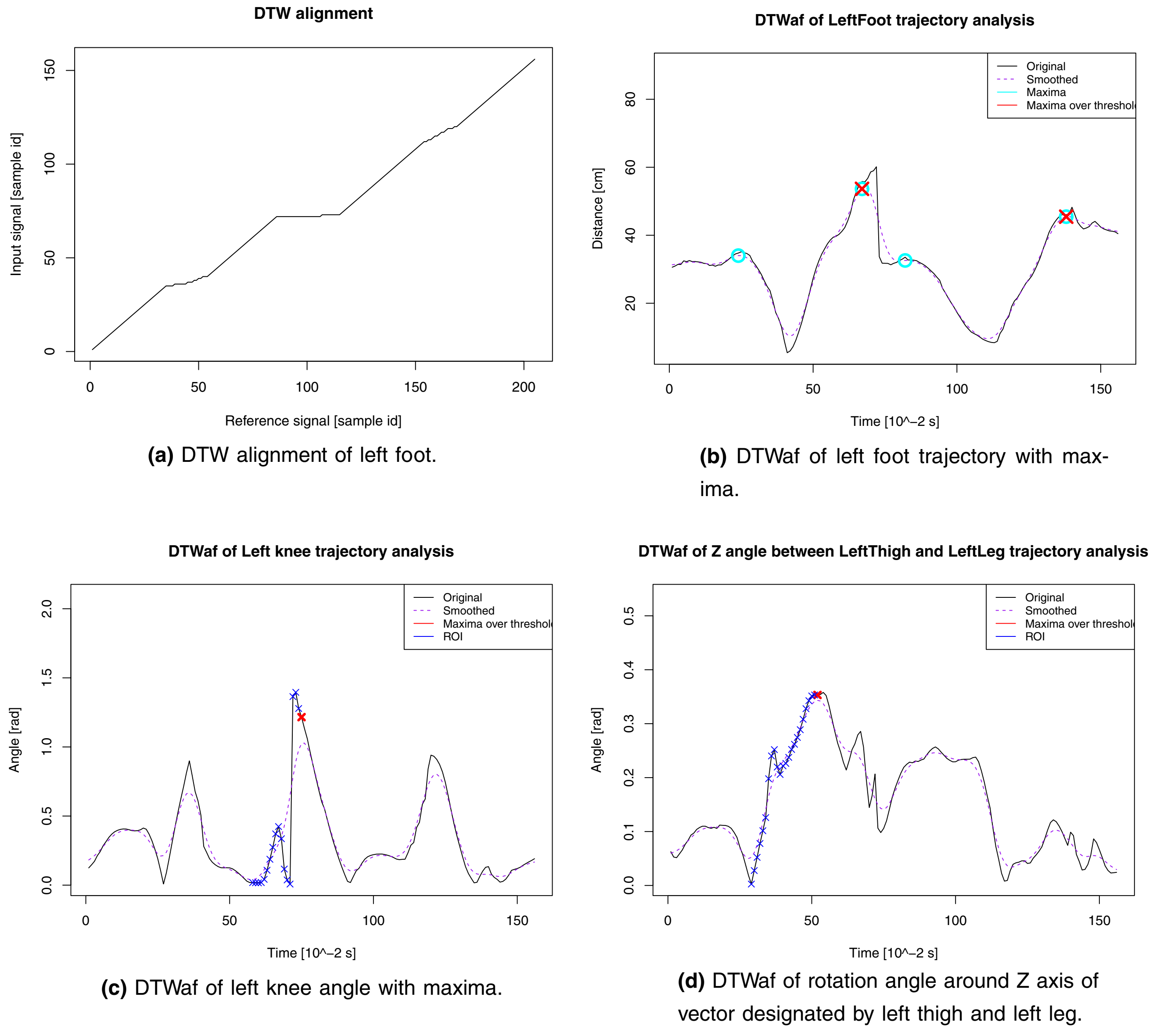

Fig. 8 This image presents the subset of plots that are generated during analysis of mawashi geri kick with left leg

This procedure is applied to each sample *i* of input MoCap.

4. We perform DTW on end joints of reference and input signal, we find DTWaf, we smooth it with Gaussian kernel and next we calculate first derivative DTWaf' using central difference numerical operator. We find local maxima in DTWaf'. Maximum is present in the moment of time *i* when DTWaf'(i) > 0 and DTWaf'(i) < 0. We take into consideration only those maxima, which have relative value above certain threshold. Example results of DTWaf analysis is presented in Fig. 8.
5. Analysis of all other joints from kinematic chain is identical. At first, we perform DTW alignment of both input and reference signals; however, we use warping paths calculated in the previous step (for end joints). We calculate, smooth, find and threshold local maxima in DTWaf' of a joint of kinematic chain using the same approach as above. We take into consideration only those maxima that are close enough to maxima detected in first step.

Figure 8 contains four sub-plots. Sub-plot (a) presents DTW alignment of left foot trajectory between reference and input signal. In this case, the left foot is the end joint and as we already mentioned distance function used by DTW is Euclidean metric. Sub-plot (b) presents DTWaf of left foot trajectory calculated according to Pseudocode 5. Original DTWaf is a solid black line, violet dashed line is DTWaf

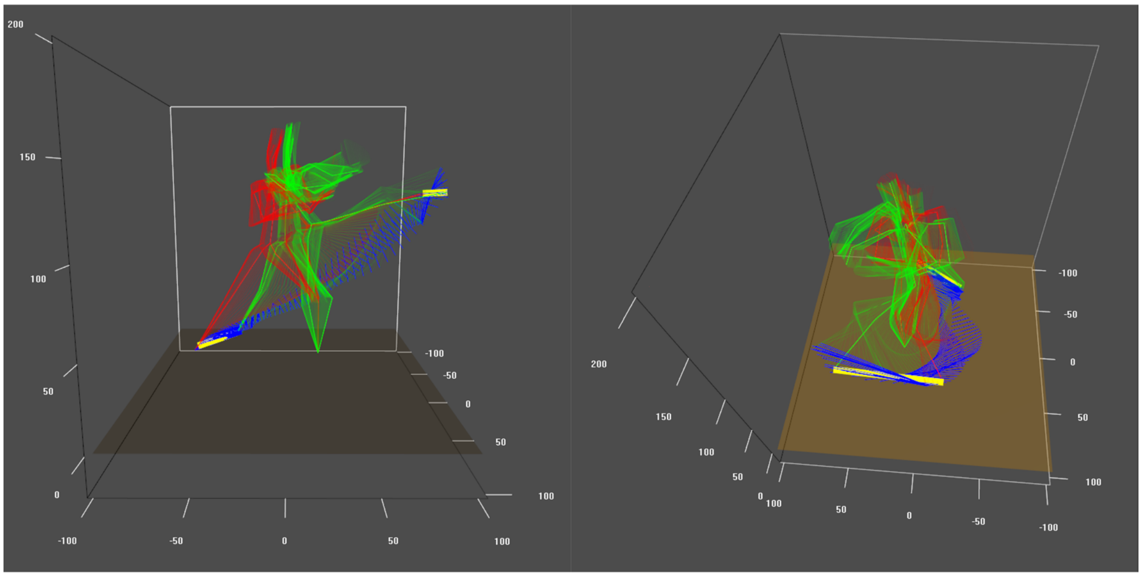

Fig. 9 3D visualization of motion analysis of mawashi geri kick with left leg. Reference data is green, input red. Blue lines indicate aligning of motion frames. Yellow line shows local maxima (color figure online)

smoothed with Gaussian kernel. Cyan circles indicate local maxima detected in smoothed signal by our algorithm. Red crosses are those maxima which values are over a threshold. As can be seen the largest difference in foot trajectory has been detected near 0.6 s and 1.4 s. Those two local maxima are indicated in Fig. 9 with yellow lines. Sub-plots (c) and (d) visualize similar analysis of DTWaf however of different body joints. In (c), DTWaf of left knee angle is visualized. This time we only show those local maxima that are within time span threshold of maxima detected in sub-plot (b) (see point 5 of motion analysis algorithm). Blue crosses preceding maxima are sample point in smoothed DTWaf in which first derivative is greater than zero. Sub-plot (d) contains visualization with analysis of DTWaf rotation angle around Z axis of vector designated by left thigh and left leg.

The motion analysis algorithm is implemented in *analyze.mocap* function. Because *R* code for data preparation and function calls look clumsy after inserting necessary line brakes to fit into this paper an example code is presented in Supplements—R replication code. In Fig. 9 we present 3D visualization of motion analysis of mawashi geri kick with left leg performed by *analyze.mocap* function.

7 Summary and discussion

Package *RMoCap* is a complete, advanced open-source tool for scientists, engineers and computer graphics who are familiar with *R* language. It provides them with MoCap data handling, statistical processing, visualizing and analysis with the help of high-level *R* programming language. *RMoCap* package uses well-established MoCap file exchange format and can be easily integrated with most of motion analysis workflows. It also introduces implementations of both novel and already known MoCap processing methods. What is more is the first *R* package that is devoted entirely for MoCap processing and analysis purposes. Those are the most important feature of our programming library.

The main drawback of *RMoCap* is long time of data processing, especially while converting from DK to HK model and data averaging. It is caused by many algebraic and optimization procedures calls that happen during calculation. However, those functions are often called only once at the beginning or end of data analysis and can be computed in the batch.

Acknowledgements This work has been supported by the National Science Centre, Poland, under project number 2015/17/D/ST6/04051.

References

1. Adler, D., Murdoch, D., et al.: RGL: 3D Visualization Using OpenGL. R package version 0.99.16 (2018)

2. Barre, A., Armand, S.: Biomechanical toolkit: open-source framework to visualize and process biomechanical data. Comput. Methods Progr. Biomed. **114**(1), 80–87 (2014)
3. Burger, B., Toiviainen, P.: Mocap toolbox—a matlab toolbox for computational analysis of movement data. In: Bresin, R. (Ed.) Proceedings of the Sound and Music Computing Conference 2013, pp. 172–178 (2013)
4. Delp, S., Anderson, F., Arnold, A., Loan, P., Habib, A., John, C., Guendelman, E., Thelen, D.: Opensim: open-source software to create and analyze dynamic simulations of movement. IEEE Trans. Biomed. Eng. **54**(11), 1940–1950 (2007)
5. Flam, D., de Queiroz, D., de Souza Ramos, T., de Albuquerque Araujo, A., Gomide, J.: Openmocap: an open source software for optical motion capture. In: 2009 VIII Brazilian Symposium on Games and Digital Entertainment, pp. 151–161 (2009)
6. Gama, J., Fuller, J., Leva, P.: RSpincalc: Conversion Between Attitude Representations of DCM, Euler Angles, Quaternions, and Euler Vectors. R package version 1.0.2 (2015)
7. Gheller, R., Pupo, J.D., Ache-Dias, J., Detanico, D., Padulo, J., dos Santos, S.: Effect of different knee starting angles on intersegmental coordination and performance in vertical jumps. Hum. Mov. Sci. **42**, 71–80 (2015)
8. Giorgino, T.: DTW: Dynamic Time Warping Algorithms. R package version 1.20-1 (2018)
9. Hachaj, T., Ogiela, M., Koptyra, K.: Application of assistive computer vision methods to oyama karate techniques recognition. Symmetry **7**(4), 1670–1698 (2015)
10. Hachaj, T., Ogiela, M.R.: Heuristic method for calculation of human body translation using data from inertial motion capture costume. Int. J. Electr. Electr. Eng. Telecommun. **1**, 26–29 (2018)
11. Hachaj, T., Ogiela, M.R.: RMocap: an R package for processing and analysing motion capture data. R package version 1.0.0 (2018)
12. Hachaj, T., Piekarczyk, M., Ogiela, M.: Human actions analysis: templates generation, matching and visualization applied to motion capture of highly-skilled karate athletes. Sensors **17**(11), 1–24 (2017)
13. Hamilton, N.: Smoother: Functions Relating to the Smoothing of Numerical Data. R package version 1.1 (2015)
14. Keane, J.: mocapGrip: An R package that encapsulates a motion capture grip and gesture analysis project. University of Chicago. R package version 0.4 (2016)
15. King, A.A., Rowan, T.: subplex: Unconstrained Optimization using the Subplex Algorithm. R package version 1.5-4 (2018)
16. Kong, P., Luk, T., Hong, Y.: Difference between Taekwondo roundhouse kick executed by the front and back leg biomechanical study. In: 18 International Symposium on Biomechanics in Sports, pp. 268–272 (2005)
17. Lee, C.L., Chin, Y.F., Liu, Y.: Comparing the difference between front-leg and back-leg round-house kicks attacking movement abilities in Taekwondo. In: 23 International Symposium on Biomechanics in Sports, pp. 877–880 (2005)
18. Ligges, U., et al.: signal: signal processing. R package version 0.7-6 (2015)
19. Markley, F., Cheng, Y., Crassidis, J., Oshman, Y.: Averaging quaternions. J. Guid. Control Dyn. **30**, 1193–1197 (2007)
20. Michael Meredith, S.M.: Motion capture file formats explained (2001)
21. Moreira, P., Goethel, M., Gonçalves, M.: Neuromuscular performance of Bandal Chagui: comparison of subelite and elite taekwondo athletes. J. Electromyogr. Kinesiol. **30**, 55–65 (2016)
22. Neto, O., Magini, M.: Electromiographic and kinematic characteristics of Kung Fu Yau-Man palm strike. J. Electromyogr. Kinesiol. **18**(6), 1047–52 (2008)
23. Ojeda, A., Bigdely-Shamlo, N., Makeig, S.: Mobilab: an open source toolbox for analysis and visualization of mobile brain/body imaging data. Front. Hum. Neurosci. **8**, 1–9 (2014)
24. Petitjean, F., Ketterlin, A., Gançarski, P.: A global averaging method for dynamic time warping, with applications to clustering. Pattern Recognit. **44**(3), 678–693 (2011)
25. Sandholm, A., Pronost, N., Thalmann, D.: Motionlab: A matlab toolbox for extracting and processing experimental motion capture data for neuromuscular simulations. pp. 110–124 (2009)
26. Sarda-Espinosa, A.: dtwclust: Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance. R package version 5.5.0 (2018)
27. Simchoni, G.: mocap: R interface to parsing and plotting Motion Capture ASF/AMC files. R package version 0.0.0.9000 (2017)
28. Singh, T., Suresh, P., Chandan, S.: Forward and inverse kinematic analysis of robotic manipulators. Int. Res. J. Eng. Technol. IRJET **4**(2), 1459–1469 (2017)
29. Soltani, P., Figueiredo, P., Fernandes, R., Vilas-Boas, J.: Do player performance, real sport experience, and gender affect movement patterns during equivalent exergame? Comput. Hum. Behav. **63**, 1–8 (2016)
30. Vishnoi, N., Mitra, A., Duric, Z., Gerber, N.: Motion based markerless gait analysis using standard events of gait and ensemble Kalman filtering. Conf Proc IEEE Eng Med Biol Soc., pp. 2512–2516 (2014)
31. Vuk, S., Markovic, G., Jaric, S.: External loading and maximum dynamic output in vertical jumping: the role of training history. Hum. Mov. Sci. **31**(1), 139–151 (2012)
32. Wieber, P., Billet, F., Boissieux, L., Pissard-Gibollet, R.: The humans toolbox, a homogenous framework for motion capture, analysis and simulation. In: Proceedings of the Ninth ISB Symposium on 3D Analysis of Human Movement, Valenciennes, France (2006)
33. Witte, K., Emmermacher, P., Bystrzycki, S., Potenberg, J.: Movement structures of round kicks in karate. In: 25 International Symposium on Biomechanics in Sports, pp. 302–305 (2007)