RMoCap package documentation

July 27, 2018

Type Package

Title Package for processing and analysis motion capture (mocap) data.

Version 1.0.0

Author Tomasz Hachaj, Marek R. Ogiela

Maintainer Tomasz Hachaj <tomekhachaj@o2.pl>

Description With functions from this package you can:
- load and save Biovision Hierarchy (BVH) files,
- convert direct kinematic model to hierarchical kinematic model and vice versa,
- plot interactive plots of motion capture (mocap) data,
- align two mocap recordings,
- while walking with translation and acceleration data is is possible to calculate correct body joints displacement,
- average many motion capture recordings that presents the same activity and store it as a single one,
- perform comparision analysis of two motion capture recordings using advanced Dynamic Time Wapring - besed procedures and display results,

This package has also many useful algebraic functions like Quaternion Markley averaging algorithms etc.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports smoother,
rgl,
RSpincalc,
subplex,
signal,
compiler

RoxygenNote 6.0.1

R topics documented:

`aligninputandrefdata` *This function translates inputdata so that it has common values of limbname Dx, Dy, Dz columns with refdata.*

Description

This function translates inputdata so that it has common values of limbname Dx, Dy, Dz columns with refdata.

Usage

```
aligninputandrefdata(inputdata, refdata, limbname)
```

Arguments

`inputdata`	input motion capture data frame.
`refdata`	reference motion capture data frame.
`limbname`	name of the column which is used to perform alignment.

Value

data frame aligned as it is described above.

Examples

```
#See example from analyze.mocap function
```

`analyze.mocap` *This function performs motion capture data analysis based on Dynamic Time Warping.*

Description

This procedure detects highest differences between reference and input mocap recording. Analysis goes as follows:

- Perform DTW on x1 and x2 from list at index 1. Plot DTW alignment This alignment will be used in aligning all other signals. Plot distance between x1 and x2 after alignment and find local maxima in this plot. In following analysis only maxima with relative value above threshold will be used.
- perform DTW alignment for each other element from the data.configuration list, however use the alignment function from first step. Maxima are detected in the same procedure as above, however we take into account only those of them, that are close enough to maxima from first step of analysis.

Usage

```
analyze.mocap(data.configuration, ref.d = NULL, in.d = NULL,
  extremumthreshold = 0.66, smoothSize = 0.1)
```

Arguments

`data.configuration`

a list containing configuration for the algorithm. Each element of the list is a list with following elements (all elements are obligatory):

- x1 - reference signal for DTW (vectors list),
- x2 - second signal for DTW (vectors list),
- FUN - distance function for DTW, use euc.dist or euc.dist1d, however you can define any function that can operates on x1 and x2,
- ylab - label on Y axis of the results plot,
- legend - part of the legend over plots,
- plotRGL - name of the body joint for which DTW alignment of x1 and x2 will be drawn. It will be 3D rgl plot. If plotRGL = NULL plot will not be drawn.

- skeleton - object of class mocap. It is used to get joints relations while plotting RGL plot.

`ref.d`	data frame with reference mocap data (default is ref.d=NULL). It is used to get joints relations while plotting RGL plot.
`in.d`	data frame with reference mocap data (default is in.d=NULL). It is used to get joints relations while plotting RGL plot.
`extremumthreshold`	threshold from range [0,1], that is used to remove local extreme, which relative value is below extremumthreshold (default value is extremumthreshold=0.66).
`smoothSize`	size of the Gaussian smoothing window. Default value is smoothSize = 0.1.

Value

a list containing data.configuration parameters plus algorihm results

Examples

```
########################
#analyze upper body data
########################
data(right.arm.motion.1)
data(right.arm.motion.2)

refdata <- right.arm.motion.1$data.frame
inputdata <- right.arm.motion.2$data.frame

extremumthreshold <- 0.66
smoothSize <- 0.1

inputdataalignment <- rotatedata(inputdata, refdata, "LeftShoulder","RightShoulder")
inputdataalignmentkinematic <- calculate.kinematic(inputdataalignment, bodypartname = "LeftShoulder")
refdatakinematic <- calculate.kinematic(refdata, bodypartname = "LeftShoulder")
inputdataalignmentkinematic <- aligninputandrefdata(inputdataalignmentkinematic, refdatakinematic, limbname

data.configuration <- list()
data.configuration[[1]] <- list(x1 = vector.to.list(refdatakinematic, "RightHand"),
   x2 = vector.to.list(inputdataalignmentkinematic, "RightHand"),
   FUN = euc.dist,
   ylab = "Distance [cm]",
   legend = "RightHand",
   plotRGL = "RightHand",
   skeleton = right.arm.motion.1)

data.configuration[[2]] <- list(x1 = vector.to.angles.list(refdatakinematic, "RightShoulder", "RightArm", "Ri
  x2 = vector.to.angles.list(inputdataalignmentkinematic, "RightShoulder", "RightArm", "RightForearm"),
   FUN = euc.dist1d,
   ylab = "Angle [rad]",
   legend = "Right elbow",
   plotRGL = NULL,
   skeleton = NULL)

x1 <- vector.to.angles.frame.list(refdatakinematic, "RightArm", "RightForearm", "RightShoulder", "LeftShould
x2 <- vector.to.angles.frame.list(inputdataalignmentkinematic, "RightArm", "RightForearm", "RightShoulder", "

data.configuration[[3]] <- list(x1 = x1[[1]],
   x2 = x2[[1]],
   FUN = euc.dist1d,
   ylab = "Angle [rad]",
   legend = "X angle between RightArm and RightForearm",
   plotRGL = NULL,
   skeleton = NULL)

data.configuration[[4]] <- list(x1 = x1[[2]],
   x2 = x2[[2]],
   FUN = euc.dist1d,
   ylab = "Angle [rad]",
   legend = "Y angle between RightArm and RightForearm",
   plotRGL = NULL,
   skeleton = NULL)

data.configuration[[5]] <- list(x1 = x1[[3]],
   x2 = x2[[3]],
   FUN = euc.dist1d,
   ylab = "Angle [rad]",
   legend = "Z angle between RightArm and RightForearm",
   plotRGL = NULL,
   skeleton = NULL)

res <- analyze.mocap(data.configuration,
refdatakinematic,
inputdataalignmentkinematic,
extremumthreshold,
smoothSize)

########################
#analyze lower body data
########################
data(mawashi.geri.left.1)
data(mawashi.geri.left.2)

refdata <- mawashi.geri.left.1$data.frame
inputdata <- mawashi.geri.left.2$data.frame
extremumthreshold <- 0.66
smoothSize <- 0.1

inputdataalignment <- rotatedata(inputdata, refdata, "LeftThigh","RightThigh")
inputdataalignmentkinematic <- calculate.kinematic(inputdataalignment, bodypartname = "RightFoot")
refdatakinematic <- calculate.kinematic(refdata, bodypartname = "RightFoot")
inputdataalignmentkinematic <- aligninputandrefdata(inputdataalignmentkinematic, refdatakinematic, limbname

data.configuration <- list()
data.configuration[[1]] <- list(x1 = vector.to.list(refdatakinematic, "LeftFoot"),
 x2 = vector.to.list(inputdataalignmentkinematic, "LeftFoot"),
 FUN = euc.dist,
 ylab = "Distance [cm]",
  legend = "LeftFoot",
  plotRGL = "LeftFoot",
  skeleton = mawashi.geri.left.1)

data.configuration[[2]] <- list(x1 = vector.to.angles.list(refdatakinematic, "LeftThigh", "LeftLeg", "LeftFoo
  x2 = vector.to.angles.list(inputdataalignmentkinematic, "LeftThigh", "LeftLeg", "LeftFoot"),
  FUN = euc.dist1d,
  ylab = "Angle [rad]",
  legend = "Left knee",
  plotRGL = NULL)

x1 <- vector.to.angles.frame.list(refdatakinematic, "LeftThigh", "LeftLeg", "LeftThigh","RightThigh")
x2 <- vector.to.angles.frame.list(inputdataalignmentkinematic, "LeftThigh", "LeftLeg", "LeftThigh","RightThi

data.configuration[[3]] <- list(x1 = x1[[1]],
  x2 = x2[[1]],
  FUN = euc.dist1d,
  ylab = "Angle [rad]",
  legend = "X angle between LeftThigh and LeftLeg",
  plotRGL = NULL)

data.configuration[[4]] <- list(x1 = x1[[2]],
  x2 = x2[[2]],
  FUN = euc.dist1d,
  ylab = "Angle [rad]",
  legend = "Y angle between LeftThigh and LeftLeg",
  plotRGL = NULL)

data.configuration[[5]] <- list(x1 = x1[[3]],
  x2 = x2[[3]],
  FUN = euc.dist1d,
  ylab = "Angle [rad]",
  legend = "Z angle between LeftThigh and LeftLeg",
  plotRGL = NULL)

res <- analyze.mocap(data.configuration,
refdatakinematic,
inputdataalignmentkinematic,
extremumthreshold,
smoothSize)
```

averaged.mocap-class *A class returned by mocap.averaging function.*

Description

A class returned by mocap.averaging function.

Usage

```
see documentation of mocap.averaging.
```

Format

a list containing data frame (fullData) and data frame (norm.distance) with normalized distance optimized during averaging.

Examples

```
data("mawashi.geri.right.list")
myList <- list()
for (a in 1:length(mawashi.geri.right.list))
{
  myList[[a]] <-mawashi.geri.right.list[[a]]$data.frame
}
res.data <- mocap.averagingCmp(myList, 2, eps = 0.000001)
plot(res.data)
```

avg.quaternion.markley

Quaternion Markley averaging algorithms.

Description

See: F. Landis Markley, Yang Cheng, John Lucas Crassidis, and Yaakov Oshman. "Averaging Quaternions", Journal of Guidance, Control, and Dynamics, Vol. 30, No. 4 (2007), pp. 1193-1197. https://doi.org/10.2514/1.28949

Usage

```
avg.quaternion.markley(Q)
```

Arguments

Q	a data frame of quaternions (four dimensional vectors) to be averaged. Each row of data frame holds one quaternion.

Value

4D quaternion vector.

Examples

```
Q <- data.frame(c(0.9999986, 0.9999986, 0.9999986, 0.9999986, 0.9999986, 0.9999986),
c(0.0008716584, 0.0008716584, 0.0009590162, 0.0009590162, 0.001046359, 0.001046359),
c(0.0009608439, 0.001048034, 0.0008736689, 0.001048034, 0.0009608439, 0.0008736689),
c(0.001046359, 0.0009590162, 0.001046359, 0.0008716584, 0.0008716584, 0.0009590162))
avg.quaternion.markley(Q)
```

`bvh.to.df` *This function get direct kinematic from list generated with function read.bvh or read.mocap function and returns it in the form of data frame.*

Description

This function does not perform any algebraic calculation - it just takes data from Dxyz and Rxyz columns. Function can additionally calculates second derivative of Dxyz.

Usage

```
bvh.to.df(skeleton, sd = TRUE)
```

Arguments

`skeleton`	input hierarchical kinematic model.
`sd`	does function should calculate second derivative? Default = TRUE.

Value

data frame with direct kinematic.

Examples

```
#an example BVH file
data("heian.nidan.bvh")
#write file to the disc
f <- file("heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in BVH file
heian.nidan <- read.bvh("heian.nidan.bvh")
df <- bvh.to.df(heian.nidan)
```

`calculate.kinematic` *This function corrects translation of the direct kinematic model.*

Description

This heuristic method is especially usable while dealing with data acquired by IMU (Inertial measurement unit) sensors. There are two possible calls of this method. The first one utilize acceleration data of body joints (.ax, .ay and .az columns). If this data is available method can calculate displacement of long motion, during which an actor uses both left and right leg. If this call is used, one should left bodypartname as NULL. The second call is used when we are dealing with data without acceleration data and actor has one stationary limb. In this case bodypartname should have value of the limb which does not translate during motion.

Usage

```
calculate.kinematic(dd, LeftFoot = "LeftFoot", RightFoot = "RightFoot",
  show.plot = FALSE, plot.title = "", bodypartname = NULL, dyEps = 5)
```

Arguments

`dd`	data frame with motion capture data.
`LeftFoot`	name of the column with data that holds coordinates of left foot (joint that touch the ground). Default value is LeftFoot = "LeftFoot". Used only for the first type of call.
`RightFoot`	name of the column with data that holds coordinates of right foot (joint that touch the ground). Default value is RightFoot = "RightFoot". Used only for the first type of call.
`show.plot`	if TRUE (default is show.plot = FALSE) plots results of an algorithm.
`plot.title`	part of the title over plots.
`bodypartname`	if not NULL value the stationary joint of the motion is column with bodypartname. Default value is bodypartname = NULL.
`dyEps`	Threshold value for translation calculation (used only for the first type of call). Default value is dyEps = 5.

Value

Data frame, which has Dxzy columns updated. All other columns are not updated.

Examples

```
 #This example uses acceleration data to calculates correct displacement.
 #header of mocap file
 data("header.mocap")
 #data frame with displacements and acceleration data
 data("heian.shodan")

heian.shodan.corrected <- calculate.kinematic(heian.shodan, show.plot = "TRUE", plot.title = "Heian Shodan")
 original.bvh <- set.data.frame(header.mocap, heian.shodan)
 corrected.bvh <- set.data.frame(header.mocap, heian.shodan.corrected)
 #plot BVH, red is original data, green is corrected
plot(original.bvh, frames.fraction = 0.1, my.color = "red", alpha = 0.1, spheres = FALSE)
plot(corrected.bvh, frames.fraction = 0.1, my.color = "green", alpha = 0.1, spheres = FALSE, append = TRUE)

 #writing BVH to disk
 write.bvh(original.bvh, "original.bvh")
 write.bvh(corrected.bvh, "corrected.bvh")

 #This example uses single body joint which coordinates should be constant during motion
 data(mawashi.geri.left.1)
plot(mawashi.geri.left.1, frames.fraction = 0.1, my.color = "red", alpha = 0.5, spheres = FALSE)
mawashi.geri.left.1$data.frame <- calculate.kinematic(mawashi.geri.left.1$data.frame, bodypartname = "Right
plot(mawashi.geri.left.1, frames.fraction = 0.1, my.color = "green", alpha = 0.5, spheres = FALSE, append = TR
```

`df.to.bvh` *This function recalculates direct to hierarchical kinematic model.*

Description

Procedure implements iterative algebraic procedure with additional initial optimization, that is required to align root body joints. Optimization is done using simplex method. The rotation order in hierarchical model is automatically set to ZYX, even if input.skeleton has different order.

Usage

```
df.to.bvh(input.skeleton, df.to.save, plot.me = FALSE, frame.id = -1,
  debug.messages = FALSE)
```

Arguments

`input.skeleton`	object of mocap class that defines hierarchical kinematic model.
`df.to.save`	data frame with column names compatible with input.skeleton. Data that is used for calculation has to be placed in columns with names ending .Dx, .Dy and .Dz.
`plot.me`	if TRUE plot steps of skeleton aligning of frame with index frame.id. Default value is plot.me = FALSE.
`frame.id`	if frame.id > 0 and plot.me = TRUE plot steps of skeleton aligning of frame with index frame.id. Default value is frame.id = -1.
`debug.messages`	print additional messages informing about calculation progress.

Value

object of class mocap.

Examples

```
 data("header.mocap")
 data("heian.yondan")

 input.skeleton <- header.mocap

 df.to.save <- heian.yondan[1:300,]
first.frame <- df.to.bvh(input.skeleton, df.to.save, plot.me = FALSE, debug.messages = TRUE)
 write.bvh(first.frame, "e:\\bvh in r\\gotowy_kod\\output\\heian.yondan.frames300.bvh")

plot(first.frame$skeleton$Joints[[1]]$Rxyz[,1], type = "l", col = "black", xlab = "sample", ylab = "angle (de
 lines(first.frame$skeleton$Joints[[1]]$Rxyz[,2], type = "l", col = "red")
 lines(first.frame$skeleton$Joints[[1]]$Rxyz[,3], type = "l", col = "blue")
legend("bottomright", legend=c("X axis rotation", "Y axis rotation", "Z axis rotation"), col=c("black", "red"
 title("Hips rotation data")

 plot(df.to.save[,2], ylab = "Displacement [cm]", xlab = "Time [10^-2 sec]", pch = 1)
 for (a in 1:ncol(df.to.save))
 {
    df.to.save[,a] <- jitter(df.to.save[,a], factor = 500)
 }
 points(df.to.save[,2],col="red", pch = 2)
```

```
legend("bottomright", legend=c("Original", "Jitter"), col=c("black", "red"), pch = c(1,2))
 title("Example channel of MoCap data")

first.frame <- df.to.bvh(input.skeleton, df.to.save, plot.me = FALSE, debug.messages = TRUE)

 #plot rotation data
plot(first.frame$skeleton$Joints[[1]]$Rxyz[,1], type = "l", col = "black", xlab = "sample", ylab = "angle (deg
 lines(first.frame$skeleton$Joints[[1]]$Rxyz[,2], type = "l", col = "red")
 lines(first.frame$skeleton$Joints[[1]]$Rxyz[,3], type = "l", col = "blue")
legend("bottomright", legend=c("X axis rotation", "Y axis rotation", "Z axis rotation"), col=c("black", "red"
 title("Hips rotation data")

write.bvh(first.frame, "e:\\bvh in r\\gotowy_kod\\output\\jitter.heian.yondan.frames300.bvh")

 df.to.save <- heian.yondan[1000:1001,]
foo <- df.to.bvh(input.skeleton, df.to.save, plot.me = TRUE, debug.messages = FALSE, frame.id = 1)
```

`euc.dist` *This function calculates Euclidean distance between vectors x1 and x2.*

Description

This function calculates Euclidean distance between vectors x1 and x2.

Usage

```
euc.dist(x1, x2)
```

Arguments

`x1`	first numeric vector.
`x2`	second numeric vector.

Value

asqrt(sum((x1 - x2) ^ 2)).

Examples

```
euc.dist(c(1,2,3,4),c(-5,0,-6, 3))
```

`euc.dist1d` *This function returns absolute value of difference between x1 and x2.*

Description

This function returns absolute value of difference between x1 and x2.

Usage

```
euc.dist1d(x1, x2)
```

Arguments

`x1`	first numeric value.
`x2`	second numeric value.

Value

abs(x1[1] - x2[1]).

Examples

```
euc.dist1d(1,-5)
```

`generate.first.frame` *This function generates object of mocap class with zero rotation data (rotation angle is set to 0).*

Description

This function generates object of mocap class with zero rotation data (rotation angle is set to 0).

Usage

```
generate.first.frame(input.skeleton, Nframes = 1, FrameTime = 0.01)
```

Arguments

`input.skeleton`	object of mocap class which defines hierarchical model.
`Nframes`	number of frames to be generated.
`FrameTime`	intervals between acquisitions.

Value

object of mocap class.

Examples

```
data("heian.nidan.bvh")
f <- file("heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in hierarchical BVH file
heian.nidan <- read.mocap("heian.nidan.bvh")
first.frame <- generate.first.frame(heian.nidan)
plot(heian.nidan, frame = 1, alpha = 1, spheres = TRUE)
plot(first.frame, my.color = "red", frame = 1, alpha = 1, spheres = TRUE, append = TRUE)
```

`header.mocap` *An example object of class mocap that does not have any motion data.*

Description

An example object of class mocap that does not have any motion data.

Usage

```
data("header.mocap")
```

Format

An object of mocap class.

`heian.nidan.bvh` *A raw bvh file to be saved on disc (Shotokan Karate kata Heian Nidan).*

Description

A raw bvh file to be saved on disc (Shotokan Karate kata Heian Nidan).

Usage

```
data(heian.nidan.bvh)
```

Format

A raw file.

Examples

```
data("heian.nidan.bvh")
f <- file("e:\\bvh in r\\gotowy_kod\\output\\heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
```

`heian.shodan` *A data frame with mocap data (Shotokan Karate kata Heian Shodan). Also acceleration channels are included.*

Description

A data frame with mocap data (Shotokan Karate kata Heian Shodan). Also acceleration channels are included.

Usage

```
data("heian.shodan")
```

Format

A data frame.

`heian.yondan` *A data frame with mocap data (Shotokan Karate kata Heian Yondan).*

Description

A data frame with mocap data (Shotokan Karate kata Heian Yondan).

Usage

```
data("heian.yondan")
```

Format

A data frame.

`hierarchical.to.direct.kinematic` *Generates direct kinematic model from hierarchical kinematic model.*

Description

This function makes calculations based on hierarchical kinematic data in input list (input.skeleton). It does not use Dxyz from input.skeleton.

Usage

```
hierarchical.to.direct.kinematic(input.skeleton)
```

Arguments

`input.skeleton`	list in the same format as one generated with read.mocap function.

Value

data frame with direct kinematic model. Names of the columns are the same as in input.skeleton.

Examples

```
#an example BVH file
data("heian.nidan.bvh")
f <- file("heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in hierarchical BVH file
heian.nidan <- read.mocap("heian.nidan.bvh")
#plot kinematic data
plot(x = heian.nidan$data.frame$Hips.Dx, y = heian.nidan$data.frame$Hips.Dz, type = "l", ylab = "Displacement X
title("Hips displacement during motion")

#generate kinematic from hierarchical model - same results as above
df <- hierarchical.to.direct.kinematic(heian.nidan$skeleton)
plot(x = df$Hips.Dx, y = df$Hips.Dz, type = "l", ylab = "Displacement X [cm]", xlab = "Displacement Z [cm]")
title("Hips displacement during motion")
```

`mawashi.geri.left.1` *An object of class mocap that contains karate kick mawashi geri.*

Description

An object of class mocap that contains karate kick mawashi geri.

Usage

```
data("mawashi.geri.left.1")
```

Format

An object of mocap class.

`mawashi.geri.left.2` *An object of class mocap that contains karate kick mawashi geri.*

Description

An object of class mocap that contains karate kick mawashi geri.

Usage

```
data("mawashi.geri.left.2")
```

Format

An object of mocap class.

`mawashi.geri.right.list` *A list of ten object of class mocap that contains karate kicks mawashi geri.*

Description

A list of ten object of class mocap that contains karate kicks mawashi geri.

Usage

```
data("mawashi.geri.right.list")
```

Format

A list of objects of mocap class.

`mocap-class` *A class returned by read.mocap function.*

Description

A class returned by read.mocap function.

Usage

```
read.mocap("file.name.bvh")
```

Format

a list containing:

- Joints - list of joints,
- Time - vector with time series,
- FrameTime - value of time interval between samples,
- Frame - samples count.

Each joint is a list that contains:

- Nestdepth - level of joint in hierarchy,
- Name - name of the joint,
- Parent - id of the parent on the list, root joint has parent = -1,
- Offset - 3D vector with offset from parent joint,
- Nchannels - number of data channels (6 from root, 3 for other or 0 for end point),
- Order - rotation order (accepted orders are XYZ, XZY, YXZ, YZX, ZXY or ZYX),
- Dxyz - matrix with direct kinematic displacement (calculated from original data),
- RawDxyz - matrix with direct kinematic displacement, present only in root joint,
- Rxyz - matrix with rotation in degrees of hierarchical kinematic model,
- Trans - list of rotation - translation matrices that are used to recalculates hierarchical to direct kinematic model.

Examples

```
#an example BVH file
data("heian.nidan.bvh")
#write file to the disc
f <- file("heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in BVH file
heian.nidan <- read.mocap("heian.nidan.bvh")
summary(heian.nidan)
```

`mocap.averaging` *This function averages a list of motion capture recordings.*

Description

Averaging is performed on each rotation channel of hierarchical model separately with Dynamic Time Warping barycentre averaging (DBA), see: François Petitjean, Alain Ketterlin, PierreGançarski, "A global averaging method for dynamic time warping, with applications to clustering", Pattern Recognition, Volume 44, Issue 3, March 2011, Pages 678-693, https://doi.org/10.1016/j.patcog.2010.09.013 Each rotation signal holds rotation represented as quaternion. Quaternion averaging is performed with Quaternion Markley averaging algorithms, see: . F. Landis Markley, Yang Cheng, John Lucas Crassidis, and Yaakov Oshman. "Averaging Quaternions", Journal of Guidance, Control, and Dynamics, Vol. 30, No. 4 (2007), pp. 1193-1197. https://doi.org/10.2514/1.28949 Results are smoothed with Weighted Quaternion Markley averaging algorithms using Gaussian kernel.

Usage

```
mocap.averaging(myList, DBAIterationsCount = 50, eps = 1e-04,
  plot.me = TRUE)
```

Arguments

`myList`	list of mocap data frames. Algorithm uses columns with names that has .Rx, .Ry and .Rz names. Rotation should be represented by Euler angles in degrees.
`DBAIterationsCount`	maximal number of iterations of DBA algorithm (default value is DBAIterationsCount = 50).
`eps`	threshold value for DBA - iteration stops when absolute value of difference between normalized DTW distances on this and previous iteration is less than eps (default value is eps = 0.0001).
`plot.me`	if TRUE, plots DTW distances for each averaged signal (default value is plot.me = 50).

Value

return object of class averaged.mocap.

Examples

```
#load list of objects of mocap class
data("mawashi.geri.right.list")
myList <- list()
#assign data frames to list
for (a in 1:length(mawashi.geri.right.list))
{
  myList[[a]] <-mawashi.geri.right.list[[a]]$data.frame
}
#run compiled version of mocap.averaging function
res.data <- mocap.averagingCmp(myList, 50, eps = 0.000001)
plot(res.data)
#write results to disc as bvh file
skel <- set.data.frame(mawashi.geri.right.list[[1]], res.data$fullData)
write.bvh(path = "avg.mawashi.geri.right.bvh", skeleton.helper = skel)
```

`myEV2Q` *Calculates quaternion from axis angle.*

Description

All other quaternion function are imported from package RSpincalc.

Usage

```
myEV2Q(axis, angle)
```

Arguments

axis	3D vector.
angle	in radians.

Value

4D quaternion vector.

Examples

```
myEV2Q(vector.to.unit(c(1,2,3)),pi/4)
```

plot.averaged.mocap *Plots normalized distance of all joints from averaged.mocap class.*

Description

Plots normalized distance of all joints from averaged.mocap class.

Usage

```
## S3 method for class 'averaged.mocap'
plot(data.to.plot)
```

Arguments

`data.to.plot`	object of class averaged.mocap.

Examples

```
data("mawashi.geri.right.list")
myList <- list()
for (a in 1:length(mawashi.geri.right.list))
{
  myList[[a]] <-mawashi.geri.right.list[[a]]$data.frame
}
res.data <- mocap.averagingCmp(myList, 2, eps = 0.000001)
plot(res.data)
```

plot.mocap *Overrides plot function to work with mocap class.*

Description

This function uses rgl package for 3D visualizations. It creates interactive 3D plots that enables rotation and scaling.

Usage

```
## S3 method for class 'mocap'
plot(obj, frame = 0, my.color = "green",
  frames.fraction = 0.1, alpha = 0.05, spheres = FALSE, append = FALSE,
  print.text = FALSE)
```

Arguments

`obj`	mocap object.
`frame`	index of frame to be show. If default (frame = 0) all frames are drawn.
`my.color`	color of the plot.
`frames.fraction`	if frame = 0 this parameter indicates what fraction of frames should be drawn (default frames.fraction = 0.1 means that 10% of frames will be plotted).

`alpha`	value of alpha channel of the plot (0 is 100% transparency, 1 is no transparency, default is alpha = 0.05).
`spheres`	if TRUE, position of body joints will be marked as spheres (default is spheres = FALSE).
`append`	if FALSE (default is append = FALSE) a new plot is generated, if TRUE a plot is drawn over open rgl window).
`print.text`	if TRUE (default is append = FALSE) plots name of the joints.

Examples

```
data("right.arm.motion.1")
plot(right.arm.motion.1, frame = 1, my.color = "white", alpha = 1, spheres = TRUE)
plot(right.arm.motion.1, frames.fraction = 0.5, my.color = "white", alpha = 1, spheres = FALSE, print.text = TRU
```

`read.bvh` *This function read file in BVH (Biovision Hierarchy) format and recalculates it to direct kinematic model.*

Description

BVH file contains a hierarchical kinematic model definition (called a skeleton) and motion data. For more information see: "Motion Capture File Formats Explained" by M. Meredith S. Maddock, doi: 10.1.1.103.2097.

Usage

```
read.bvh(filepath)
```

Arguments

`filepath`	A path to BVH file.

Value

an object of mocap class.

Examples

```
#an example BVH file
data("heian.nidan.bvh")
#write file to the disc
f <- file("heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in BVH file
```

`read.mocap` *This function reads motion capture file on BVH format.*

Description

It also calculates direct kinematic from hierarchical kinematic. This function calls read.bvh and bvh.to.df to generate single object of mocap class. See documentation for those two functions.

Usage

```
read.mocap(filepath)
```

Arguments

`filepath`	path to a file.

Value

object of mocap class.

Examples

```
#an example BVH file
data("heian.nidan.bvh")
f <- file("eian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in hierarchical BVH file
heian.nidan <- read.mocap("heian.nidan.bvh")
summary(heian.nidan)
```

`right.arm.motion.2` *An object of class mocap that contains motion of right arm.*

Description

An object of class mocap that contains motion of right arm.

Usage

```
data("right.arm.motion.2")
```

Format

An object of mocap class.

`rotatedata` *This function align two data frames that contains mocap data.*

Description

Both data frames need to have two common column groups with names ending Dx and Dz. This function calculates vector vv = v1-v2 for both data frames (vv.m for mydata and vv.ref for referencedata) and rotates mydata around Y axis in order to minimize euclidean distance between vv.m and vv.ref. Minimization is made with simplex method. After this procedure mydata face the same direction as referencdata. This procedure works correctly only if root joint of mocap is stationary.

Usage

```
rotatedata(mydata, referencedata, v1, v2)
```

Arguments

`mydata`	input data frame with mocap data to be algined to referencedata.
`referencedata`	reference data frame with mocap data.
`v1`	name of the first body joint.
`v2`	name of the second body joint.

Value

mydata rotated by Y axis so that mydata and referencedata faces same direction.

Examples

```
data(mawashi.geri.left.1)
data(mawashi.geri.left.2)
refdata <- mawashi.geri.left.1$data.frame
inputdata <- mawashi.geri.left.2$data.frame
#after following function inputdata and refdata are alignined towards vector LeftThigh - RightThigh
inputdataalignment <- rotatedata(inputdata, refdata, "LeftThigh","RightThigh")
```

`rotation.matrix.between.vectors` *This function returns n by n rotation matrix that align vector x onto y.*

Description

Both vector x and y has to be normalized.

Usage

```
rotation.matrix.between.vectors(x, y)
```

Arguments

`x`	vector to be rotated (has to be normalized).
`y`	reference vector (has to be normalized).

Value

n x n rotation matrix.

Examples

```
x <- vector.to.unit(c(1,0,0))
y <- vector.to.unit(c(0,1,0))
Rx2y <- rotation.matrix.between.vectors(x, y)
x %*% Rx2y
y
x <- vector.to.unit(c(5,0,2,4,6))
y <- vector.to.unit(c(0,1,0,8,-5))
Rx2y <- rotation.matrix.between.vectors(x, y)
x %*% Rx2y
y
x <- vector.to.unit(c(8,0,0))
y <- vector.to.unit(c(2,0,0))
Rx2y <- rotation.matrix.between.vectors(x, y)
x %*% Rx2y
y
```

`second.derivative` *Calculates second derivative*

Description

Calculates second derivative using Central Difference Operator.

Usage

```
second.derivative(signal)
```

Arguments

`signal`	input vecotr.

Value

vector containing second derivative calculated from the input vector.

Examples

```
second.derivative(sin(seq(from=0,to=2*pi,by=pi/50)))
```

set.data.frame — *Assign data frame to object of mocap class.*

Description

This function does not recalculate hierarchical kinematic model in skeleton.

Usage

```
set.data.frame(skel, df)
```

Arguments

skel	object of mocap class.
df	data frame with columns names compatible to hierarchical model definition in skel object.

Value

object of mocap class with df assigned.

Examples

```
data("header.mocap")
data("heian.shodan")
print(header.mocap$skeleton$Frames)
original.bvh <- set.data.frame(header.mocap, heian.shodan)
print(original.bvh$skeleton$Frames)
```

summary.mocap — *Plots information about number of frames, joints count and joints hierarchy of mocap class.*

Description

Plots information about number of frames, joints count and joints hierarchy of mocap class.

Usage

```
## S3 method for class 'mocap'
summary(mocap.data)
```

Arguments

mocap.data	an object of mocap class.

Examples

```
data("heian.nidan.bvh")
f <- file("e:\\bvh in r\\gotowy_kod\\output\\heian.nidan.bvh")
writeChar(con = f, object = heian.nidan.bvh)
close(f)
#read hierarchical model stored in hierarchical BVH file
heian.nidan <- read.mocap("e:\\bvh in r\\gotowy_kod\\output\\heian.nidan.bvh")
summary(heian.nidan)
```

`vector.angle` *This function calculates angle between two vectors on the plane designated by those vectors.*

Description

The return value is in the range [0,pi].

Usage

```
vector.angle(a, b)
```

Arguments

a	first vector.
b	second vector.

Value

angle between a and b (in radians).

Examples

```
vector.angle(c(1,2,3),c(4,5,6))
```

`vector.cross` *This function calculates cross product.*

Description

Cross product is only defined for 3D vectors.

Usage

```
vector.cross(a, b)
```

Arguments

a	first vector.
b	second vector.

Value

cross product.

Examples

```
vector.cross(c(1,2,3),c(3,4,5))
```

`vector.dot` *This function calculates dot product.*

Description

This function calculates dot product.

Usage

```
vector.dot(a, b)
```

Arguments

a	first vector.
b	second vector.

Value

dot product.

Examples

```
vector.dot(c(1,2,3,4),c(5,6,7,8))
```

`vector.norm` *This function returns length (norm) of the vector.*

Description

This function returns length (norm) of the vector.

Usage

```
vector.norm(x)
```

Arguments

x	a vector.

Value

vector norm.

Examples

```
vector.norm(c(1,2,3))
```

`vector.to.angles.frame.list` *This function generates list of angles between vector and coordinates frame.*

Description

Vector is defined as v1=(f1-f2). The coordinate frame is:

- X=(xt1-xt2),
- Z=X x (0,1,0),
- Y=X x Z

All vectors are normalized.

Usage

```
vector.to.angles.frame.list(dataToCalculate, f1, f2, xt1, xt2)
```

Arguments

`dataToCalculate`	data frame with motion capture data.
`f1`	name of the first joint of vector.
`f2`	name of the second joint of vector.
`xt1`	name of the first joint of horizontal axis of coordinate frame.
`xt2`	name of the second joint of horizontal axis of coordinate frame.

Value

Results are list of vectors defined as follow: (angle(Y, v1), angle(Z, v1), angle(X, v1)).

Examples

```
data("heian.yondan")
vector.to.angles.frame.list(heian.yondan[1:3,], "RightArm", "RightForearm", "RightShoulder", "LeftShoulder")
```

`vector.to.angles.list` *This function generates list of angles.*

Description

Angles are calculated on the plane between vectors v1=(f2-f1) and v2=(f2-f3).

Usage

```
vector.to.angles.list(dataToCalculate, f1, f2, f3)
```

Arguments

`dataToCalculate`	data frame with motion capture data.
`f1`	name of the first joint.
`f2`	name of the second joint.
`f3`	name of the third joint.

Value

list of angles (in radians) defined as above.

Examples

```
data("heian.yondan")
vector.to.angles.list(heian.yondan[1:3,], "RightShoulder", "RightArm", "RightForearm")
```

`vector.to.list` *This function generates list of vectors.*

Description

Each list position equals c(ABC.Dx, ABC.Dy, ABC.Dz) where featuresName = ABC is a name of the column of data frame dataToCalculate.

Usage

```
vector.to.list(dataToCalculate, featuresName)
```

Arguments

`dataToCalculate`	data frame with motion capture data.
`featuresName`	column name of the motion capture feature. Column names should have names endings .Dx, .Dy, .Dz.

Value

list of vectors defined as above.

Examples

```
data("heian.yondan")
vector.to.list(heian.yondan[1:3,], "RightHand")
```

`vector.to.unit` *This function performs vector normalizing.*

Description

This function performs vector normalizing.

Usage

```
vector.to.unit(x)
```

Arguments

`x`	a vector.

Value

normalized vector x.

Examples

```
vector.to.unit(c(1,2,3,4,5))
```

`wavg.quaternion.markley` *Weighted quaternion Markley averaging algorithms.*

Description

See: F. Landis Markley, Yang Cheng, John Lucas Crassidis, and Yaakov Oshman. "Averaging Quaternions", Journal of Guidance, Control, and Dynamics, Vol. 30, No. 4 (2007), pp. 1193-1197. https://doi.org/10.2514/1.28949

Usage

```
wavg.quaternion.markley(Q, weights)
```

Arguments

`Q`	a data frame of quaternions (four dimensional vectors) to be averaged. Each row of data frame holds one quaternion.
`weights`	weights vector.

Value

4D quaternion vector.

Examples

```
Q <- data.frame(c(0.9999986, 0.9999986, 0.9999986, 0.9999986, 0.9999986, 0.9999986),
c(0.0008716584, 0.0008716584, 0.0009590162, 0.0009590162, 0.001046359, 0.001046359),
c(0.0009608439, 0.001048034, 0.0008736689, 0.001048034, 0.0009608439, 0.0008736689),
c(0.001046359, 0.0009590162, 0.001046359, 0.0008716584, 0.0008716584, 0.0009590162))

x <- seq(-2,2,length=6)
y <- dnorm(x,mean=0, sd=1)
y <- y / sum(y)
wavg.quaternion.markley(Q, y)
```

`write.bvh` *This function saves object of mocap class to a file.*

Description

Function uses hierarchical model definition from hierarchy list however data of channels are taken from data frame.

Usage

```
write.bvh(skeleton.helper, path)
```

Arguments

`skeleton.helper`	mocap object to be save.
`path`	path to the file.

Examples

```
 data("header.mocap")
 data("heian.shodan")
heian.shodan.corrected <- calculate.kinematic(heian.shodan, show.plot = "TRUE", plot.title = "Heian Shodan")
 original.bvh <- set.data.frame(header.mocap, heian.shodan)
 corrected.bvh <- set.data.frame(header.mocap, heian.shodan.corrected)
 #plotting BVH
plot(original.bvh, frames.fraction = 0.1, my.color = "red", alpha = 0.1, spheres = FALSE)
plot(corrected.bvh, frames.fraction = 0.1, my.color = "green", alpha = 0.1, spheres = FALSE, append = TRUE)
 #writing BVH to disk
 write.bvh(original.bvh, "original.bvh")
 write.bvh(corrected.bvh, "corrected.bvh")
```

Index