<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>4</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>4</td>
</tr>
<tr>
<td>Release Notes</td>
<td>5</td>
</tr>
<tr>
<td><strong>Version 1.1</strong></td>
<td>5</td>
</tr>
<tr>
<td>- New features</td>
<td>5</td>
</tr>
<tr>
<td>- Bug fixes</td>
<td>5</td>
</tr>
<tr>
<td><strong>Version 1.2</strong></td>
<td>5</td>
</tr>
<tr>
<td>- New features</td>
<td>5</td>
</tr>
<tr>
<td>- Bug fixes</td>
<td>6</td>
</tr>
<tr>
<td><strong>Version 1.2.1</strong></td>
<td>6</td>
</tr>
<tr>
<td>- Bug fixes</td>
<td>6</td>
</tr>
<tr>
<td><strong>Version 1.2.2</strong></td>
<td>6</td>
</tr>
<tr>
<td>- Bug fixes</td>
<td>6</td>
</tr>
<tr>
<td><strong>Version 1.3</strong></td>
<td>7</td>
</tr>
<tr>
<td>- New features</td>
<td>7</td>
</tr>
<tr>
<td>- Bug fixes</td>
<td>7</td>
</tr>
<tr>
<td>Introduction</td>
<td>9</td>
</tr>
<tr>
<td>General</td>
<td>11</td>
</tr>
<tr>
<td>Functions</td>
<td>12</td>
</tr>
<tr>
<td>Data Structures</td>
<td>15</td>
</tr>
<tr>
<td>Parameter structures</td>
<td>16</td>
</tr>
<tr>
<td>Examples</td>
<td>18</td>
</tr>
<tr>
<td>Reading, Editing, and Visualizing MoCap Data (mcdemo1)</td>
<td>19</td>
</tr>
<tr>
<td>Transforming MoCap data (mcdemo2)</td>
<td>28</td>
</tr>
<tr>
<td>Kinematic analysis (mcdemo3)</td>
<td>32</td>
</tr>
<tr>
<td>Time-series analysis (mcdemo4)</td>
<td>38</td>
</tr>
</tbody>
</table>
Kinetic analysis (mcdemo5) .........................................................41
Creating animations (mcdemo6 - mcdemo9) .................................46
Basics (mcdemo6)............................................................................46
Merging data for animations (mcdemo7) .........................................47
Colored animations (mcdemo8) ......................................................48
Perspective Projection (mcdemo9) ...................................................51
Principal Components Analysis (mcdemo10) ................................53
Analyzing Wii data (mcdemo11) ....................................................54

Data and Parameter Structure Reference 58
MoCap data structure ...................................................................59
norm data structure ....................................................................60
segm data structure ...................................................................61
m2jpar parameter structure ............................................................62
j2spar parameter structure ............................................................63
animpar parameter structure ..........................................................64

Function Reference 65
mc2frontal .................................................................................66
mcanimate ..................................................................................67
mcbandpass ................................................................................68
mcboundrect ..............................................................................69
mccenter ....................................................................................70
mccreateconnmatrix ..................................................................71
mcumdist ...................................................................................72
mccut .........................................................................................73
mcdecompose .............................................................................74
mc-eigenmovement .....................................................................75
mcfillgaps ..................................................................................76
mcfilteremg ..............................................................................77
mcgetmarker ..............................................................................78
mcgetmarkernname ....................................................................79
mcgetsegmpar ............................................................................80
mchilbert ...................................................................................81
mchilberthuang .........................................................................82
mcicaproj ..................................................................................83
mcinitanimpar ............................................................................84
mcinitj2spar ..............................................................................86
mcinitm2jpar .............................................................................87
mcinitstruct ..............................................................................88
mcj2s .........................................................................................89
mckinenergy .............................................................................90
mckurtosis ................................................................................91
mcm2j .......................................................................................93
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcmarkerdist</td>
<td>94</td>
</tr>
<tr>
<td>mcmean</td>
<td>95</td>
</tr>
<tr>
<td>mcmerge</td>
<td>96</td>
</tr>
<tr>
<td>mcmissing</td>
<td>97</td>
</tr>
<tr>
<td>mcnorm</td>
<td>98</td>
</tr>
<tr>
<td>mcpcaproj</td>
<td>99</td>
</tr>
<tr>
<td>mcperiod</td>
<td>100</td>
</tr>
<tr>
<td>mcpotenergy</td>
<td>105</td>
</tr>
<tr>
<td>mcread</td>
<td>106</td>
</tr>
<tr>
<td>mcreademg</td>
<td>107</td>
</tr>
<tr>
<td>mcreorderdims</td>
<td>108</td>
</tr>
<tr>
<td>mcresample</td>
<td>109</td>
</tr>
<tr>
<td>mcrotate</td>
<td>110</td>
</tr>
<tr>
<td>mcs2j</td>
<td>111</td>
</tr>
<tr>
<td>mcs2posture</td>
<td>112</td>
</tr>
<tr>
<td>mcsegmangle</td>
<td>113</td>
</tr>
<tr>
<td>mcsethares</td>
<td>114</td>
</tr>
<tr>
<td>mcsetmarker</td>
<td>115</td>
</tr>
<tr>
<td>mcsimmat</td>
<td>116</td>
</tr>
<tr>
<td>mcskewness</td>
<td>117</td>
</tr>
<tr>
<td>mcsmoother</td>
<td>118</td>
</tr>
<tr>
<td>mcspectrum</td>
<td>119</td>
</tr>
<tr>
<td>mcstatmoments</td>
<td>120</td>
</tr>
<tr>
<td>mcstd</td>
<td>121</td>
</tr>
<tr>
<td>mctimeader</td>
<td>122</td>
</tr>
<tr>
<td>mctimeintegr</td>
<td>124</td>
</tr>
<tr>
<td>mctranslate</td>
<td>125</td>
</tr>
<tr>
<td>mctrim</td>
<td>126</td>
</tr>
<tr>
<td>mcvar</td>
<td>127</td>
</tr>
<tr>
<td>mcvect2grid</td>
<td>128</td>
</tr>
<tr>
<td>mcwindow</td>
<td>129</td>
</tr>
</tbody>
</table>
This manual provides an introduction and a reference to the MoCap Toolbox, a Matlab® toolbox for the analysis and visualization of Motion Capture data. The toolbox is mainly aimed for the analysis of music-related movement, but might be useful in other areas of study as well. I wrote most of the toolbox and this manual during my sabbatical at the Center for Advanced Study in the Behavioral Sciences at Stanford University in 2007-8.

This manual requires that the user be familiar with the basic features of the Matlab software. Novices in this programming platform are advised to consult the cornucopia of Matlab tutorials available on the Internet. The reader should also be familiar with the basics of mechanics and calculus.

I would like to thank the Academy of Finland and the Center for Advanced Study in the Behavioral Sciences at Stanford University for their support.

Stanford, June 1, 2008

Petri Toiviainen

Acknowledgments

Thanks to JJ Loh for providing a faster version for reading in .c3d files.

Thanks to Erwin Schoonderwaldt for providing the function to read in .mat files (exported from QTM).

Thanks to Michiel Demey, Frank Desmet, Alexander Refsum Jensenius, Luiz Naveda, and Erwin Schoonderwaldt for reporting bugs in the toolbox.
Release Notes

Version 1.1

New features

mcbandpass: band pass filter MoCap data
mceigenmovement: generates eigenmovements from PCA
mchilbert: performs a Hilbert transform
mcicaproj: performs an ICA on MoCap data
mcpcaproj: performs a PCA on MoCap data
mcs2j: performs a segment-to-joint mapping
mcs2posture: creates a posture representation form segm data
mcsethares_mb: performs an m-best Sethares transform
mcsethares_s2l: performs a small-to-large Sethares transform
mcimamat: calculates similarity matrix
mcvect2grid: converts MoCap structure vector to MoCap structure with three orthogonal views

PCA example added in the Examples section

Bug fixes

mcicaproj: fix a bug to allow to handle cases when not all ica's converge
mcmissing: allow to use norm data as input

Version 1.2

New features

mccreateconnmatrix: creates a connection matrix using QTM label list files (for plotting)
mchilberthuang: performs a Hilbert-Huang transform
mcreademg: read in emg data (Mega EMG system, recorded into QTM)
mcsethares: combines mcsethares_mb and mcsethares_s2l in one function
mcvar: calculates the variance of (MoCap) data
mcdemo8: demo added on how to use colors in plotting and animations
mcdemo9: demo added on how to use PCA on movement data

**Bug fixes**
mcbandpass: add gaussian weighting as a method
mcfillgaps: fill gaps in the beginning and end with the first/last recorded values
mcgetmarker: allow to use norm data as input
mcplotframe, mcinitanimpar: individual colors for markers, traces, connectors, and numbers for plotting and animations
mcread: handling of the various .tsv export options of QTM and faster .c3d import (including marker names as well)
mcresample: allow to use norm data as input

---

**Version 1.2.1**

**Bug fixes**
mcplotframe: plotting frame numbers, no stats toolbox for animations
mcmerge: concatenates marker and trace color strings (if both animations shall have the same colors)

---

**Version 1.2.2**

**Bug fixes**
mcfillgaps: filling of begin and end with first or last recorded marker ('fillall') fixed. New option ('nobefill') that fills gaps during recording, but gaps in start and end are set to NaN instead of 0
mcm2j: number of markers set correctly in the joint representation
mcplottimeseries: frame scale when plotting norm data
mcplottimeseries: plot more than one time series in the same plot using different colors

mcplotframe: frame numbers issue fixed

mc2frontal: change marker direction to left to right

**Version 1.3**

**New features**

mcboundrect: calculates the bounding rectangle

mccut: cuts two MoCap structures to the length of the shorter one

mcdemo9: demo on how to use perspective projection in animations

mcfilteremg: filters emg data

mcinitstruct: initializes mocap or norm structure

mcreadmat: reads in .mat files as exported from QTM

mcsegmangle: calculates the angles between two markers

mcsetmarker: replaces a subset of markers in an existing mocap or norm structure

mcspectrum: Calculates the amplitude spectrum of mocap time series

**Bug fixes**

mcanimate, mcplotframe, mcinitanimpar: perspective projection implemented

mccumdist: also for data that has already norm structure

mcfillgaps: bug fixed regarding empty markers/trajectories and bug fixed on input parameter

mcmerge: now also for norm data

mcperiod: option for either first or highest peak value of autocorrelation

mcplotframe: png output format when creating animations

mcplottimeseries: plotting more than one time series in the same plot for norm data

mcread: opportunity to read in .mat files as exported from QTM

mcreadc3d: number of frames issue solved (though see note for mcread)
mcreadwii: can handle 6-dof data (order: X Y Z Roll Pitch Yaw)
mcrate: bug fixed regarding missing markers
mcsmoother: Butterworth filter smoothing implemented
mctimeder (1): polynomial order and window length dependent on the order of the derivative
mctimeder (2): implementation of faster version based on differences of successive frames
mcwindow: extended for mcvar
Introduction
The MoCap Toolbox is a Matlab® toolbox that contains functions for the analysis and visualization of motion capture data. It supports the generic .c3d file format, the .tsv data format produced by the Qualisys motion capture system, the .mat file format produced by the Qualisys motion capture system, and the .wii format produced by the WiiDataCapture software (available at www.jyu.fi/music/coe/materials/mocaptoolbox).

To use the toolbox, you need the Matlab software (www.mathworks.com). Before using it, the toolbox has to be added in the Matlab path variable. The toolbox should be compatible with most versions of Matlab and most platforms. It has been tested on Matlab versions 7.4-7.11 running on Macintosh OS X v10.4-10.6.

To use all the functions in the MoCap Toolbox, the following toolboxes must be included in Matlab's path:

- Signal Processing Toolbox
- Periodicity Toolbox, available at http://eceserv0.ece.wisc.edu/~sethares/downloadper.html (for mcsethares)

Register to the MoCap Toolbox mailing list: www.jyu.fi/music/coe/materials/mocaptoolbox to stay informed about new releases, bug reports, and bug fixes. It also serves as a general discussion board for users, so feel free to post anything motion capture- and toolbox-related that might be of interest to other users and developers. The email address to send messages to the list is mocaptoolbox@freelists.org (requires registration to send).

The MoCap Toolbox comes with no warranty. It is free software, and you are welcome to redistribute it under certain conditions. See the file License.txt provided with the toolbox for details of GNU General Public License.
Functions

The MoCap Toolbox contains 60 functions for the analysis and visualization of motion capture data. The functions can be divided into nine categories:

- Data input and edit functions
- Coordinate transformation functions
- Coordinate system conversion functions
- Kinematic analysis functions
- Kinetic analysis functions
- Time-series analysis functions
- Visualization functions
- Projection functions
- Other functions

The following table provides an overview of the functions available in the MoCap Toolbox. Detailed descriptions of each function are provided in the Chapter Function Reference.

<table>
<thead>
<tr>
<th>INPUT &amp; EDIT</th>
<th>Function</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcread</td>
<td>read MoCap data files</td>
<td></td>
</tr>
<tr>
<td>mcreademg</td>
<td>read emg files in tsv format</td>
<td></td>
</tr>
<tr>
<td>mcmissing</td>
<td>report missing frames and markers</td>
<td></td>
</tr>
<tr>
<td>mctrim</td>
<td>extract a temporal segment from MoCap data</td>
<td></td>
</tr>
<tr>
<td>mccut</td>
<td>cut two MoCap structures to the length of the shorter one</td>
<td></td>
</tr>
<tr>
<td>mcsmoothen</td>
<td>smoothen MoCap data</td>
<td></td>
</tr>
<tr>
<td>mcmerge</td>
<td>merge two MoCap data structures</td>
<td></td>
</tr>
<tr>
<td>mcgetmarker</td>
<td>extract a subset of markers from MoCap data</td>
<td></td>
</tr>
<tr>
<td>mcsetmarker</td>
<td>replace a subset of markers</td>
<td></td>
</tr>
<tr>
<td>mcgetmarkernamex</td>
<td>get names of markers from MoCap data</td>
<td></td>
</tr>
<tr>
<td>mcfillgaps</td>
<td>fill missing data</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>mcinitstruct</td>
<td>initialize MoCap or norm data structure</td>
<td></td>
</tr>
<tr>
<td>mcreorderdims</td>
<td>reorder the Euclidean dimensions in the MoCap data</td>
<td></td>
</tr>
<tr>
<td>mcresample</td>
<td>resample motion capture data</td>
<td></td>
</tr>
<tr>
<td>mccenter</td>
<td>center MoCap data to have a mean of [0 0 0]</td>
<td></td>
</tr>
<tr>
<td>mctranslate</td>
<td>translate MoCap data</td>
<td></td>
</tr>
<tr>
<td>mcrotate</td>
<td>rotate MoCap data</td>
<td></td>
</tr>
<tr>
<td>mc2frontal</td>
<td>rotate MoCap data to have a frontal view with respect to a pair of markers</td>
<td></td>
</tr>
<tr>
<td>mcvect2grid</td>
<td>convert a MoCap structure vector to a MoCap structure with three orthogonal views</td>
<td></td>
</tr>
<tr>
<td>mcinitm2jpar</td>
<td>initialize parameters for marker-to-joint mapping</td>
<td></td>
</tr>
<tr>
<td>mcm2j</td>
<td>perform a marker-to-joint mapping</td>
<td></td>
</tr>
<tr>
<td>mcinitj2spar</td>
<td>initialize parameters for joint-to-segment mapping</td>
<td></td>
</tr>
<tr>
<td>mcj2s</td>
<td>perform a joint-to-segment mapping</td>
<td></td>
</tr>
<tr>
<td>mcs2j</td>
<td>perform a segment-to-joint mapping</td>
<td></td>
</tr>
<tr>
<td>mcs2posture</td>
<td>create a posture representation form segm data</td>
<td></td>
</tr>
<tr>
<td>mcnorm</td>
<td>calculate the norms of Euclidean MoCap data</td>
<td></td>
</tr>
<tr>
<td>mctimeder</td>
<td>estimate time derivatives of MoCap data</td>
<td></td>
</tr>
<tr>
<td>mctimeintegr</td>
<td>estimate time integrals of MoCap data</td>
<td></td>
</tr>
<tr>
<td>mccumdist</td>
<td>calculate the cumulative distance traveled by each marker</td>
<td></td>
</tr>
<tr>
<td>mcmarkerdist</td>
<td>calculate the distance of a marker pair</td>
<td></td>
</tr>
<tr>
<td>mcboundrect</td>
<td>calculate the bounding rectangle</td>
<td></td>
</tr>
<tr>
<td>mcsegmangle</td>
<td>calculate the angles between two markers</td>
<td></td>
</tr>
<tr>
<td>mcperiod</td>
<td>estimate period of MoCap data</td>
<td></td>
</tr>
<tr>
<td>mcdecompose</td>
<td>decompose kinematic variable into tangential and normal components</td>
<td></td>
</tr>
<tr>
<td>mcspectrum</td>
<td>calculate amplitude spectrum of MoCap data</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>KINETIC ANALYSIS</td>
<td>mcgetsegmpar</td>
<td>return segment parameters of a body model</td>
</tr>
<tr>
<td></td>
<td>mckinenergy</td>
<td>estimate instantaneous kinetic energy of the body</td>
</tr>
<tr>
<td></td>
<td>mcpotenergy</td>
<td>estimate instantaneous potential energy of the body</td>
</tr>
<tr>
<td>TIME-SERIES ANALYSIS</td>
<td>mcmean</td>
<td>calculate mean of MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcstd</td>
<td>calculate std of MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcvar</td>
<td>calculate variance of MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcskewness</td>
<td>calculate skewness of MoCap data</td>
</tr>
<tr>
<td></td>
<td>mckurtosis</td>
<td>calculate kurtosis of MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcstatmoments</td>
<td>calculate first four statistical moments</td>
</tr>
<tr>
<td></td>
<td>mcwindow</td>
<td>perform windowed time-series analysis</td>
</tr>
<tr>
<td>VISUALIZATION</td>
<td>mcplottimeseries</td>
<td>plot time series data</td>
</tr>
<tr>
<td></td>
<td>mcpplotphaseplane</td>
<td>create phase plane plot</td>
</tr>
<tr>
<td></td>
<td>mcinitanimpar</td>
<td>initialize animation parameters</td>
</tr>
<tr>
<td></td>
<td>mccreateconnmatrix</td>
<td>create connection matrix for plotting and animations</td>
</tr>
<tr>
<td></td>
<td>mcpplotframe</td>
<td>plot jpeg frames from MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcanimate</td>
<td>plot and save jpeg frames for animation</td>
</tr>
<tr>
<td>PROJECTION</td>
<td>mcpcaproj</td>
<td>perform a PCA on MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcicaproj</td>
<td>perform an ICA on MoCap data</td>
</tr>
<tr>
<td></td>
<td>mcsethares</td>
<td>perform either an m-best or a small-to-large Sethares transform</td>
</tr>
<tr>
<td></td>
<td>mceigenmovement</td>
<td>generate eigenmovements from PCA</td>
</tr>
<tr>
<td>OTHER</td>
<td>mcbandpass</td>
<td>band pass filter MoCap data</td>
</tr>
<tr>
<td></td>
<td>mchilbert</td>
<td>perform a Hilbert transform</td>
</tr>
<tr>
<td></td>
<td>mchilberthuang</td>
<td>perform a Hilbert-Huang transform</td>
</tr>
<tr>
<td></td>
<td>mcfilteremg</td>
<td>filter emg data</td>
</tr>
<tr>
<td></td>
<td>mcsimmat</td>
<td>calculate similarity matrix</td>
</tr>
</tbody>
</table>
Data Structures

The MoCap Toolbox uses three kinds of data structures: the MoCap data structure, the norm data structure and the segm data structure. An instance of a MoCap data structure is created by the function \texttt{mcread}, when motion capture data is read from a file to the workspace. The MoCap data structure contains the recorded locations of the markers as well as some basic information, such as the name of the file from which the data were read, the number of frames in the data, the number of cameras used in the capture session, the number of markers in the data, the sampling frequency, or frame rate, of the data, the names of the markers. The \texttt{.data} field of the MoCap data structure is a matrix containing the locations of the markers. It has three columns for each marker, corresponding to the two horizontal dimensions (1st and 2nd column) and the vertical dimension (3rd column). For instance, column 1 contains the x coordinates of marker 1, column 2 contains the y coordinates of marker 1, column 3 contains the z (vertical) coordinates of marker 1, column 4 contains the x coordinates of marker 2 etc. Each row in the matrix corresponds to a frame. Additionally, the MoCap data structure contains a field that indicates the order of time differentiation of the data, with zero corresponding to location, one to velocity, two to acceleration etc. The value of this field is changed by the functions \texttt{mctimeder} and \texttt{mctimeintegr} that perform temporal derivation and integration, respectively. Finally, the MoCap data structure contains fields that can hold data captured from analog devices, such as EEG, GSR, etc.

An instance of a MoCap data structure is also created when the function \texttt{mcmm2j} is called. This function performs a transformation from a marker representation to a joint. While these two representations use the same data structure, they are conceptually different in the sense that the marker representation is related to actual marker locations, whereas the joint representation is related to locations derived from marker locations. This representation is helpful when we wish to calculate the location of a body part where it is impossible to attach a marker. For instance the midpoint of a joint can be derived as the centroid of four markers located around the joint.

The norm data structure, created by the function \texttt{mcnorm}, is similar to the MoCap data structure, except that its \texttt{.data} field contains only one column per marker. This column holds the Euclidean norm of the vector data from which it was derived. If, for instance, the function \texttt{mcnorm} is applied to velocity data, the resulting norm data structure holds the magnitudes of velocities, or speeds, of each marker.
While the MoCap and norm data structures are related to points in space (markers or joints), the segm data structure contains data about segments of the body. The function mcj2s, which carries out a transformation from a joint representation to a segment representation, produces as output an instance of the segm data structure. The segm data structure contains most of the fields of the MoCap data structure. The .data field of the MoCap data structure is however replaced by a few other fields. The .parent field contains information about the kinematic chains of the body, in other words, how the joints are connected to each others to form segments, and how these segments are connected to each other. The location and orientation of the center of the body (the root) is contained in the fields .roottrans and .rootrot. The .segm field contains several subfields that store the orientation of the body segments in various forms. The .eucl subfield contains for each segment the euclidean vector pointing from proximal to distal joint of the segment. The .r subfield contains the length of each segment. The .quat subfield contains the rotation of each segment as a quaternion representation (to learn about the use of quaternions to represent 3D rotations, see for instance http://en.wikipedia.org/wiki/Quaternion). Finally, the .angle subfield holds the angle between each segment and its proximal segment.

A more detailed description of the data structures used in the MoCap Toolbox can be found in the Chapter Data and Parameter Structure Reference.

Parameter structures

To facilitate the converting between different representations (marker, joint, and segment) and the producing of certain visualizations, the MoCap Toolbox uses three different parameter structures: the m2jpar, j2spar, and animpar structures.

The m2jpar structure is used by the function mcm2j and contains information needed to carry out a transformation from a marker representation to a joint representation. Among other things, it contains a field that holds, for each joint, the numbers of the markers whose centroid defines the location of that joint.

The j2spar structure is used by the function mcj2s and contains information needed to carry out a transformation from a joint representation to a segment representation. Among other things, it contains the number of the joint that is the center (root) of the body. It also contains the numbers of three joints that define the frontal plane of the body. Finally, it contains a vector that indicates the
parent segment (the segment proximal in the kinematic chain) for each segment and a cell array with the segment names.

The *animpar structure* is used by the functions `mcplotframe` and `mcanimate`, and holds information needed to create frame plots and animations. These include, for both frame plots and animation, the limits of the plotted area, screen size, viewing angle, plotting colors, marker size, line widths, and connections between markers. Additionally, the structure holds the possibility to plot traces and the frames per second used when creating animations.

A more detailed description of the parameter structures used in the MoCap Toolbox can be found in the Chapter *Data and Parameter Structure Reference*.
Examples
This chapter contains a number of examples that are intended to serve as an introduction to the use of the MoCap Toolbox. The examples illustrate how MoCap data can be read into Matlab, edited, visualized, and transformed. They also explain how kinematic, kinetic, as well as time-series analysis can be performed and how animations can be created. Finally the chapter contains an example of the analysis and visualization of data captured using the Nintendo Wii remote controller.

The examples presented in this chapter can also be found in the function mcdemo. Just type mcdemo in the Matlab Command Window and follow the instructions.

**Reading, Editing, and Visualizing MoCap Data (mcdemo1)**

This example shows how you can read MoCap files into Matlab as well as how you can edit and visualize the data. Motion capture data can be imported into Matlab and stored as a MoCap data structure using the function mcread. Currently the function supports the generic .c3d format, the .tsv produced by the Qualisys ProReflex system, and the .wii file format, produced by the WiiData-Capture software (available at www.jyu.fi/music/coe/materials/mocaptoolbox).

The MoCap toolbox folder includes the .mat file mcdemodata.mat that contains motion capture data and associated parameter structures as Matlab variables. These data are used in the examples of this manual. The commands used in the demo files are marked in dark green.

```matlab
load mcdemodata
whos
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>dance1</td>
<td>1x1</td>
<td>1013572</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>dance2</td>
<td>1x1</td>
<td>1013572</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>j2spar</td>
<td>1x1</td>
<td>2168</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>japar</td>
<td>1x1</td>
<td>2530</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>m2jpar</td>
<td>1x1</td>
<td>3464</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>mapar</td>
<td>1x1</td>
<td>2914</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>walk1</td>
<td>1x1</td>
<td>241862</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>walk2</td>
<td>1x1</td>
<td>275474</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>wiiData</td>
<td>1x1</td>
<td>45560</td>
<td>struct</td>
<td></td>
</tr>
</tbody>
</table>

Variable walk1 is a MoCap data structure:

```matlab
walk1
walk1 =
```
type: 'MoCap data'
filename: '28-Karolien-Walking.tsv'
nFrames: 351
nCameras: 8
nMarkers: 28
freq: 60
nAnalog: 0
anaFreq: 0
timederOrder: 0

markerName: [28x1 cell]
data: [351x84 double]
analogdata: []

other: [1x1 struct]

Let us look if there are any missing data in the variable walk1:

```matlab
[mf, mm, mgrid] = mcmissing(walk1);
figure, set(gcf,'Position',[40 200 560 420])
subplot(3,1,1), bar(mf), xlabel('Marker'), ylabel('Num. of Missing frames')
subplot(3,1,2), bar(mm), xlabel('Frame'), ylabel('Num. of Missing markers')
subplot(3,1,3), imagesc(-mgrid'), colormap gray, xlabel('Frame'), ylabel('Marker')
```

Markers 2 and 6 have missing frames. The missing data can be filled using the function `mcfillgaps`:

```matlab
walk1 = mcfillgaps(walk1, 100);
```
The variable `walk1` has no more missing frames.

Marker location data can be plotted as a function of time using the function `mcplottimeseries`:

```matlab
mcplottimeseries(walk1, 3, 3, 'sec') % marker 3, dimension 3
mcplottimeseries(walk1, [4 8 12], 3, 'sec') % markers 4,8, & 12, dimension 3
```
mcplottimeseries(walk1, [4 8 12], 1:3, 'sec')

% markers 4, 8, & 12, dimensions 1:3

mcplotframe(walk1, 160);

Marker locations in single frames can be plotted using the function `mcplotframe`. This function plots the (x,z) projection of the markers:
Because the parameter structure was not given in the previous call, the function used the default settings for the animation parameters:

```matlab
mcinitanimpar
ans =
    type: 'animpar'
    scrszie: [800 600]
    limits: []
        az: 0
        el: 0
    msize: 12
    colors: 'kwwww'
    marker_colors: []
    conn_colors: []
    trace_colors: []
    number_colors: []
    cwidth: 1
    twidth: 1
    conn: []
    conn2: []
    trm: []
    trl: 0
    shownum: 0
    numbers: []
```
To obtain a visualisation that is easier to understand, the markers should be connected. The variable `mapar` contains, among other things, the connection matrix:

```matlab
mapar
mapar =
    type: 'animpar'
    scrsize: [400 300]
    limits: []
    az: 0
    el: 0
    msize: 6
    colors: 'kwwww'
    markercolors: []
    conncolors: []
    tracecolors: []
    numbercolors: []
    cwidth: 1
    twidth: 1
    conn: [43x2 double]
    conn2: []
    trm: []
    trl: 0
    showmnum: 0
    numbers: []
    showfnum: 0
    animate: 0
    fps: 30
    folder: 'tmp'
    pers: [1x1 struct]
```
The connection matrix is in the field .conn:

```
mapar.conn'
ans =
Columns 1 through 10
    1     2     3     3     5     9    10    11    11     8
    2     4     4     1     6    10    12    12     9     9
Columns 11 through 20
    8     8     8     5     5     6     6     7     7     7
   10     5     6     9    11    10    12    11    12     5
Columns 21 through 30
    7     5    13    13    16    15     6    14    14    17
    6    13    15    16    19    19    14    17    18    20
Columns 31 through 40
   18     9    11    10    12    21    23    25    26    22
   20    21    21    22    22    23    25    26    23    24
Columns 41 through 43
    24     9    11    10    12    21    23    25    26    22
   27    28    24
```

It also has a smaller screen size than the default:

```
mapar.scrsize
ans =
    400    300
```

For the purpose of making the frame plots look nicer on this manual, let us increase the \frame size:

```
mapar.scrsize = [800 600];
```

Using the parameter structure mapar, we get the following visualization:

```
mcplotframe(walk1, 160, mapar);
```
We can add marker numbers to the plot by setting the field `.showmnum` to have the value 1:

```matlab
mapar.showmnum = 1;
mplotframe(walk1, 160, mapar);
```

Different colors can be used by changing the value of the field `.colors`:

```matlab
mapar.colors='wbgcr';
mplotframe(walk1, 160, mapar);
```

Changing individual marker, connector, and number colors is explained in section Colored animations.
The connector widths and marker sizes can be adjusted by changing the values of the fields `cwidth` and `msize`, respectively:

```
mapar.cwidth = 3;
mapar.msize = 6;
mcpplotframe(walk1, 160, mapar);
```

The viewing azimuth and elevation can be changed by changing the values of the fields `az` and `el`, respectively:

```
mapar.az = 45;
mapar.el = 20;
mcpplotframe(walk1, 160, mapar);
```
Transforming MoCap data *(mcdemo2)*

This example shows how you can do various coordinate transformations to MoCap data and merge data collected at different motion capture sessions. Let us plot a motion-capture frame:

```matlab
load mcdemodata
mapar.scrsize=[800 600];
mapar.colors = 'wkkkk';
mcplotframe(dance1, 50, mapar);
```

Let us next rotate the data contained in the variable `dance1` by 90 degrees counterclockwise around the z (vertical) axis and plot the same frame:
Next, let us rotate the data in \texttt{dance1} by 90 degrees counterclockwise around the x axis:

\begin{verbatim}
d1rot2 = mcrotate(dance1, 90, [1 0 0]);
mcplotframe(d1rot2, 50, mapar);
\end{verbatim}

Finally, let us rotate the data in \texttt{dance1} by 90 degrees counterclockwise around the y axis:

\begin{verbatim}
d1rot3 = mcrotate(dance1, 90, [0 1 0]);
mcplotframe(d1rot3, 50, mapar);
\end{verbatim}
To add data from several MoCap data structures to one visualization, the functions `mctranslate` and `mcmerge` are useful:

```matlab
all = dance1;
% translate 'd1rot1' 2000 mm to the right and merge with 'all'
% merge also the parameter structures
[all, allparams] = mcmerge(all, mctranslate(d1rot1, [2000 0 0]),...
    mapar, mapar);
% Same with 'd1rot2' and 'd1rot3', but with different translations
[all, allparams] = mcmerge(all, mctranslate(d1rot2, [0 0 2000]),...
    allparams, mapar);
[all, allparams] = mcmerge(all, mctranslate(d1rot3, [2000 0 2000]),...
    allparams, mapar);
```

Next, let us plot one frame from the merged data:

```matlab
allparams.msize=6;
mcplotframe(all, 50, allparams);
```
Let us now take excerpts from the variables `dance1` and `dance2` and merge them for visualization

\[
\begin{align*}
d1 &= \text{mctrim}(\text{dance1}, 0, 2); \\
d2 &= \text{mctrim}(\text{dance2}, 0, 2); \\
d2 &= \text{mctranslate}(d2, [2000 \ 0 \ 0]); \\
[d, \text{par}] &= \text{mcmerge}(d1, d2, \text{mapar}, \text{mapar}); \\
\text{mcplotframe}(d, 60, \text{par});
\end{align*}
\]

Several frames can be plotted with one command by using a vector as the second parameter

\[
\text{mcplotframe}(d, 1:10:71, \text{par});
\]
(In Matlab the frames will be plotted as separate figures.)

Chapter *Creating Animations* explains how these frames can be used to create an animation.

**Kinematic analysis (mcdemo3)**

This example shows how you can estimate kinematic variables from MoCap data and visualize them.

Time derivatives of motion-capture data can be estimated using the function `mctimeder`:

```matlab
load mcdemodata
d2v = mctimeder(dance2, 1); % velocity
d2a = mctimeder(dance2, 2); % acceleration
```

Let us have a look at the vertical velocity component of markers 1, 19, and 25 (left front head, left hand, and left foot):

```matlab
mcplottimeseries(d2v, [1 19 25], 3)
```
The subplots display the vertical velocities of markers 1, 19, and 25, from top to bottom.

Next, let us plot the vertical acceleration components of the same markers:

```
mcplottimeseries(d2a, [1 19 25], 3)
```

The phase plane plots for velocity and acceleration can be produced as follows:

```
figure, set(gcf,'Position', [40 40 200 800])
% change the shape of the figure to make the subplots rectangular
mcplotphaseplane(d2v, d2a, [1 19 25], 3)
```

This figure is below on the left.

The same phase plane plot, but for the interval between 5 and 7 seconds can be produced as follows:
mcplotphaseplane(mctrim(d2v,5,7), mctrim(d2a,5,7), [1 19 25], 3)

This figure is below on the right.

The cumulative distance travelled by a marker can be calculated with the function `mccumdist`.

\[
d2dist=mccumdist(dance2);
\]

Let us have a look at the distance travelled by markers 1, 19, and 25 (left front head, left hand, and left foot):

\[
mcplottimeseries(d2dist, [1 19 25])
\]
As we can see, the head has travelled ca. 13 meters, the hand ca. 33 meters, and the foot ca. 11 meters.

Periodicity of movement can be estimated using the function `mcperiod`. Let us estimate the periodicity of the movement of marker 1 (left front head) in the three dimensions.

```matlab
d2m1 = mcgetmarker(dance2, 1);
[per, ac, eac, lag] = mcperiod(d2m1, 2); % maximal period = 2 sec
```

```
per
per =
    NaN        NaN    0.5167
```

There is thus no periodic movement along the horizontal dimensions (dimensions 1 and 2), but a period of 0.51 seconds in the vertical direction.

The autocorrelation function for the vertical location of marker 1 looks like this:

```matlab
plot(lag, ac(:,3)), xlabel('Period / secs')
```
The first maximum at non-zero lag can be found at 0.51 secs, corresponding to the previous result.

The enhanced autocorrelation function for the same data looks like this:

```matlab
plot(lag, eac(:,3)), xlabel('Period / secs')
```

Again, there is a clear maximum at the period of 0.51 secs.

More accurate periodicity analysis can be done using windowed autocorrelation:

```matlab
[per, ac, eac, lags, wtime] = mcwindow(@mcperiod, d2m1, 2, 0.25);
```

Let us plot the periodicity estimates for the vertical dimension for each of the windows

```matlab
plot(wtime, per(:,3))
xlabel('Time / secs')
ylabel('Period /secs')
```
After displaying some initial transients, the period settles at the vicinity of 0.5 secs.

The enhanced autocorrelation matrix can be plotted as an image to allow visual inspection of the time development of periodicity. The colors provide an indication of the regularity of periodic movement, with warm colors corresponding to regions of highly regular periodic movement.

```matlab
gca('XTick',0:4:46)
gca('XTickLabel',0.5*(0:4:46))
gca('YTick',[0 30 60 90 120])
gca('YTickLabel',[0 0.5 1 1.5 2.0])
xlabel('Time / secs')
ylabel('Period / secs')
```
Time-series analysis *(mcdemo4)*

This example shows how you can perform various statistical analyses on time-series data using the functions provided in the MoCap toolbox.

The first statistical moments, mean, standard deviation, skewness, and kurtosis, can be calculated using the functions `mcmean`, `mcstd`, `mcskewness`, and `mckurtosis`, respectively. These functions ignore eventual missing frames. The function `mcstatmoments` can be used to calculate these statistical moments with one function call.

Standard deviations provides a measure for the extent of movement. Let us calculate the standard deviations for the markers 1, 19, and 25 (left front head, left hand, and left foot) in the *MoCap data structures* `dance1` and `dance2`:

```matlab
load mcdemodata
std1 = mcstd(mcgetmarker(dance1, [1 19 25]));
std2 = mcstd(mcgetmarker(dance2, [1 19 25]));
figure, set(gcf,'Position',[40 200 560 420])
subplot(2,1,1)
bar(reshape(std1,3,3)), xlabel('Dimension')
legend('Head', 'Hand', 'Foot'), axis([-Inf Inf 0 400])
title('dance1')
subplot(2,1,2)
bar(reshape(std2,3,3)), xlabel('Dimension')
legend('Head', 'Hand', 'Foot'), axis([-Inf Inf 0 400])
title('dance2')
```
The standard deviations for the dimensions 1 and 2 are larger for dance1 than for dance2, suggesting that dancer 1 occupies a larger area horizontally. The standard deviation for dimension 3 for the hand marker is larger for dance1, suggesting that dancer 1 uses larger vertical hand movements than dancer 2.

Let us calculate and plot the skewness values for the vertical dimension of selected markers in variables dance1, dance2, walk1 and walk2.

```
marker = [1 9 19 21 25];
d1skew = mcskewness(mcgetmarker(dance1, marker));
w1skew = mcskewness(mcgetmarker(walk1, marker));
d2skew = mcskewness(mcgetmarker(dance2, marker));
w2skew = mcskewness(mcgetmarker(walk2, marker));
mn = mcgetmarkername(dance1);
subplot(2,2,1)
bar(d1skew(3:3:end)), set(gca,'XTickLabel', [mn{marker}])
title('dance1'), axis([-Inf Inf -2 3])
subplot(2,2,2)
bar(d2skew(3:3:end)), set(gca,'XTickLabel', [mn{marker}])
title('dance2'), axis([-Inf Inf -2 3])
subplot(2,2,3)
bar(w1skew(3:3:end)), set(gca,'XTickLabel', [mn{marker}])
title('walk1'), axis([-Inf Inf -2 3])
subplot(2,2,4)
bar(w2skew(3:3:end)), set(gca,'XTickLabel', [mn{marker}])
title('walk2'), axis([-Inf Inf -2 3])
```
There are some differences between dancing and walking with respect to the skewness values. The interpretation of these differences will be left to the user.

Windowed analysis of the statistical time-series descriptors can be carried out using the `mcwindow` command. Let us compute the windowed standard deviation of markers 1, 19, and 25 (left front head, left hand, and left foot) in the variable `dance1`:

```matlab
marker = [1 19 25];
d1std = mcwindow(@mcstd, mcgetmarker(dance1, marker), 2, 0.25);
for k=1:9
    subplot(3,3,k), plot(d1std(:,k)),
    title(['Marker ' num2str(marker(1+floor((k-1)/3)))... '
', 'dim. ' num2str(1+rem(k-1,3))])
end
```
High values in these graph correspond to temporal regions where the particular marker shows wide movements for the respective dimension.

**Kinetic analysis** *(mcdemo5)*

This example shows how the toolbox can be used to calculate kinetic variables from MoCap data using Dempster's body-segment model.

Let us estimate various forms of mechanical energy in walking movement (variable `walk1`). To start with, we plot a MoCap frame with marker numbers:

```matlab
load mcdemodata
mapar.colors = 'wkkkk';
mapar.showmnum = 1;
mapar.msize=6;
mapar.az=90;
mcplotframe(walk2,160, mapar);
```

The first thing to do is to reduce the set of markers to make the data compatible with Dempster's model. This can be accomplished using the marker-to-joint transformation, implemented in function `mcm2j`. The parameters needed for this conversion are in the variable `m2jpar`:

```matlab
m2jpar
m2jpar =
```
The information concerning which markers correspond to each joint is contained in the field `m2jpar.markerNum`. The names of the new joints are in `m2jpar.markerName`. For instance, ...

```matlab
m2jpar.markerName{1}
ans =
root

m2jpar.markerNum{1}
ans =
9    10    11    12
```

... the joint 'root' is obtained by calculating the centroid of markers 9, 10, 11 and 12.

The marker-to-joint conversion is carried out as follows:

```matlab
walk2j = mcm2j(walk2, m2jpar)
```

```matlab
walk2j =
    type: 'MoCap data'
    filename: '25-Walking.tsv'
    nFrames: 401
    nCameras: 8
    nMarkers: 20
    freq: 60
    nAnalog: 0
    anaFreq: 0
    timerDelayOrder: 0
    markerName: {1x20 cell}
        data: [401x60 double]
    analogdata: []
    other: [1x1 struct]
```

The parameters for the visualization of the joint representation are in the variable `japar`:

```matlab
japar
```
Let us visualize a frame from the new variable `walk2j` with marker numbers visible:

```matlab
japar.colors = 'wkkkk';
japar.scrsize = [800 600];
japar.showmnum = 1;
japar.msize=6;
japar.az=90;
mcplotframe(walk2j,160, japar);
```
The next step is to make a joint-to-segment transformation. The parameters needed for the transformation are in the variable \( j2spar \):

\[
j2spar = \\
\begin{align*}
type & : 'j2spar' \\
rootMarker & : 1 \\
frontalPlane & : [6 2 10] \\
parent & : [0 1 2 3 4 1 6 7 8 1 10 11 13 14 15 11 17 18 19] \\
segmentName & : \{1x19 \text{ cell}\}
\end{align*}
\]

The transformation can be accomplished using the function \( mcj2s \):

\[
walk2s = mcj2s(walk2j, j2spar)
\]

The parameters for each body segment can be obtained using the function \( mcgetsegmpar \):

\[
\begin{align*}
\text{segmindex} & = [0 0 8 7 6 0 8 7 6 13 12 10 11 3 2 1 11 3 2 1]; \\
\text{spar} & = mcgetsegmpar('Dempster', \text{segmindex});
\end{align*}
\]

The second argument in the function call, \( \text{segmindex} \), associates each body segment in \( \text{walk1j} \) and \( \text{walk1s} \) with a segment type. The numbers refer to the distal joint of the respective segment. Segment number values for model 'Dempster' are as follows: no parameter=0, hand=1, forearm=2, upper arm=3, forearm and hand=4, upper extremity=5, foot=6, leg=7, thigh=8, lower extremity=9, head=10, shoulder=11, thorax=12, abdomen=13, pelvis=14, thorax and abdomen=15, abdomen and pelvis=16, trunk=17, head, arms and trunk (to glenohumeral joint)=18, head, arms and trunk (to mid-rib)=19. For instance, the third component, 8, tells that the body segment whose distal joint is joint number 3, is a 'thigh'.

\[
\text{MoCap Toolbox Manual} \\
44
\]
Now that we have a body-segment representation of the movement, we can estimate various kinetic variables. The potential energy for each body segment can now be calculated as follows:

\[
pot = mcpotenergy(walk2j, walk2s, spar);
\]

The resulting variable `pot` is a matrix where each column corresponds to one of the body segments. Let us plot the total potential energy as a function of time:

```matlab
time = (1:walk2.nFrames)/walk2.freq;
plot(time, sum(pot,2))
xlabel('Time / s')
ylabel('Potential energy / W')
```

We can see a relatively regularly oscillating pattern, with the exception of the region between 3.0 and 4.5 seconds, where the walker turns around.

The translational and rotational energy for each body segment can be calculated as follows:

```matlab
[trans, rot] = mckinenergy(walk2j, walk2s, spar);
```

Let us plot the total translational energy:

```matlab
plot(time, sum(trans,2))
xlabel('Time / s')
ylabel('Translational energy / W')
```
We can observe a region of low translational energy in the region between 3.0 and 4.5 seconds, where the walker is turning.

**Creating animations** *(mcdemo6 - mcdemo9)*

**Basics** *(mcdemo6)*

This example shows how you can create animations with the MoCap toolbox.

You can only produce the animation frames with the MoCap toolbox. These have to be compiled into a movie using some other software. On a Macintosh you can use, for instance, the QuickTime Pro software. Should you happen to use the Windows operating system, you can use, for instance the Movie Maker software.

Let us create an animation from the variable `walk2`. The `animpar` structure `mapar` contains the connector information for this variable:

```matlab
load mcdemodata
mapar
mapar =
    type: 'animpar'
    scrsize: [400 300]
    limits: []
    az: 0
    el: 0
```
Let us change the frames-per-second value to 15

\[
\text{mapar.fps} = 15;
\]

The animation frames will be stored into the folder `tmp` that will be created in the current directory. If needed, the current directory should be changed before creating the animation.

The animation frames can be produced as follows:

\[
\text{newpar} = \text{mcanimate(walk2, mapar)};
\]

After the function has finished execution, the next thing to do is to launch a video editing program and read the frames into a video clip.

**Merging data for animations** (mcdemo7)

The next example shows how MoCap data from different sessions can be combined into the same animation. It also shows how the viewing angle can be changed dynamically. Let us create a 10-second animation with two dancers and a dynamically moving viewing angle. The variable `dance1` has some missing frames, so we shall fill them first.
dance1 = mcfillgaps(dance1);

Next, we extract the first ten seconds from the variables dance1 and dance2:

d1 = mctrim(dance1, 0, 10);
d2 = mctrim(dance2, 0, 10);

We make the viewing azimuth change dynamically from zero to 180 degrees and the elevation angle from 45 to -45 degrees during the animation:

mapar.az = [0 180];
mapar.el = [45 -45];

We set the movie to have 15 frames per second and give a name to the frame folder:

mapar.fps = 15;
mapar.folder = 'twodancers';

The next step is to translate the data in d2 by two meters to the right and merge this with the data in d1:

[d, par] = mcmerge(d1, mctranslate(d2, [2000 0 0]), mapar, mapar);

Next, if needed, we change the current directory. Now we are ready to create the animation frames.

newpar = mcanimate(d, par);

As with the previous example, the next thing to do is to launch a video editing program and read the frames into a video clip.

**Colored animations (mcdemo8)**

This example shows how to color plots and animations.

We will create a colored animation from the variable dance2. The animpar structure mapar looks like this:

```matlab
load mcdemodata
mapar

mapar =

type: 'animpar'
scrsize: [400 300]
limits: []
  az: 0
  el: 0
  msize: 6
```
colors: 'kwwww'
markercolors: []
conncolors: []
tracecolors: []
numbercolors: []
cwidth: 1
twidth: 1
conn: [43x2 double]
conn2: []
trm: []
trl: 0
shownum: 0
numbers: []
showfnum: 0
animate: 0
fps: 30
folder: 'tmp'
pers: [1x1 struct]

Let us change the frames-per-second value to 15

    mapar.fps = 15;

Let us set individual colors for six markers (head front left, head back right, shoulder left, hip left back, finger right, knee left, knee right, heel left)

    mapar.markercolors='bwwgwrwwwwwywwwwwwmwcbwg';

and let us have a look at the new colors:

    mcplotframe(dance2, 150, mapar);
Now let us set the markers that we want to trace and the trace length (in seconds):

\[ \text{mapar.trm} = [1 \ 6 \ 12 \ 19 \ 21 \ 24]; \]
\[ \text{mapar.trl} = 3; \]

And let us set individual colors for the traces:

\[ \text{mapar.tracecolors} = 'grymcb'; \]

We rotate the figure to be frontal on average

\[ \text{dance2=mc2frontal(dance2,9,10)}; \]

Now we make the animation:

\[ \text{newpar = mcanimate(dance2, mapar)}; \]

For plotting a figure with traces, just select one frame that was calculated during the animation, for instance frame number 100 of \text{dance2}:
Perspective Projection \textit{(mcdemo9)}

Next, the possibility of creating an animation with a perspective (three-dimensional) effect will be explained. We will create a couple of animations of the walk2 data, with and without the perspective projection to see the differences. Let us load the \textit{mcdemodata} and change a couple of parameters of the animpar structure \textit{mapar}.

```plaintext
load mcdemodata
mapar.scrsize=[600 400];
mapar.msize=8;
mapar.fps=15;
mapar.colors='wkkkk';
```

And we also set the azimuth parameter in the animpar structure \textit{mapar}, so that the walker will walk towards us.

```plaintext
mapar.az=270;
```

Now we create an animation out of that.

```plaintext
mcanimate(walk2, mapar);
```

If we have a look at the animation frames, the figure appears to walk on the spot, although she is actually walking forwards. As an example, have a look at the 10th, 20th, and 40th frame:

The idea of the perspective projection is to make this visible also in the current view of the walker. To activate the perspective rotation, we call the \textit{mcanimate} function again, but with one additional parameter:

```plaintext
mcanimate(walk2, mapar, 1);
```
This animation looks far more natural regarding the movement direction of the walker.

The parameters for the perspective projection are set in the `pers` part of the `animpar` structure `mapar`.

```plaintext
mapar.pers
mapar.pers =
c: [0  -4000  0]
th: [0  0  0]
e: [0  -2000  0]
```

The field `c` sets the 3D position of the camera, `th` is the orientation of the camera, and `e` stores the viewer's position relative to the display surface. We can now change, for example, the camera position and create another animation to see what happens.

```plaintext
mapar.pers.c=[1000  -4000  1000];
mcanimate(walk2, mapar, 1);
```
**Principal Components Analysis** *(mcdemo10)*

Principal components analysis can be used to decompose Motion Capture data into components that are orthogonal to each other.

Let us extract the first four seconds from the structure `dance2`,

```matlab
load mcdemodata
d=mctrim(dance2,0,4);
```

and convert it into a joint representation

```matlab
j=mcm2j(d, m2jpar);
```

Next, we calculate the first three principal component projections of the structure `j`,

```matlab
[pc,p]=mcpcaproj(j,1:3);
```

and plot the amount of variance contained in these principal components.

```matlab
bar(p.l(1:3))
```

We see that the first PC contains ca. 85% of the variance, while the next two components contain only 9% and 2%.

The PC projections can be investigated, for instance, by creating animations, for instance,

```matlab
mcanimate(pc(1), japar);
```

The animations show that the principal component distort the body segment relations, in particular, the lengths of certain body segments vary. Often better results can be obtained by performing the PCA on the segment representation

```matlab
s=mcj2s(j,j2spar); % convert to segment structure
```
Next, let us plot the first three PC projections:

\[
\text{plot(ps.c(1:3,:)'), legend('PC1','PC2','PC3')}\]

The plot reveals that the first PC correspond to non-periodic motion, while PCs 2 and 3 correspond to (almost) periodic motion. Animation of \( pcj(k) \) shows that the first three PCs correspond to translation of the body, periodic anti-phase movement of arms, and periodic rotation of torso.

**Analyzing Wii data (mcdemo11)**

The Nintendo Wiimote provides an inexpensive means for simple motion capture. This example shows how movement data collected with the Nintendo Wii controller can be analyzed using the MoCap Toolbox.

The MoCap Toolbox supports the file format used by the WiiDataCapture software, available at [www.jyu.fi/music/coe/materials](http://www.jyu.fi/music/coe/materials).

In the file `mcdemodata`, the variable `wiidata` contains acceleration data captured using the Nintendo Wii controller and the WiiDataCapture software:

\[
\text{load mcdemodata}
\]

\[
\text{wiidata}
\]
wiidata =
    type: 'MoCap data'
    filename: 'data.wii'
    nFrames: 1826
    nCameras: []
    nMarkers: 1
    freq: 100
    nAnalog: 0
    anaFreq: []
    timerOrder: 2
    markerName: {}
    data: [1826x3 double]
    analogdata: []
    other: []

As the field timerOrder indicates, this variable holds acceleration data. Let us plot the third (vertical) component of the acceleration data:

mcplottimeseries(wiidata,1,3)

The data is somewhat noisy, so we smoothen it a bit:

wd2 = mcsmoothen(wiidata,25);
mcplottimeseries(wd2,1,3)
Let us do a windowed analysis of the period of this acceleration component using a window length of two seconds and a hop factor of 0.25:

$$[\text{per, ac, eac, lags, wstart}] = \text{mcwindow}(\text{mcperiod, wd2, 2, 0.25});$$

Next, let us plot the estimated period of the third component as a function of the starting point of the window:

```matlab
plot(wstart, per(:,3))
set(gcf, 'Position', [40 200 560 420])
xlabel('Time / s')
ylabel('Period / s')
```

We observe a periodic motion that starts with a period of ca. 0.2 seconds, slows down to a period of ca. 1 second, and speeds again up to a period of ca. 0.2 seconds.
A similar representation can be obtained by plotting the enhanced autocorrelation image:

```matlab
imagesc(eac(:,:,3)), axis xy
set(gcf,'Position', [40 200 560 420])
set(gca,'XTick',0:2:32)
set(gca,'XTickLabel',0.5*(0:2:32))
set(gca,'YTick',[1 51 101 151 201])
set(gca,'YTickLabel',[0 0.5 1 1.5 2.0])
xlabel('Time / secs')
ylabel('Period / secs')
```
Data and Parameter Structure Reference
MoCap data structure

**synopsis**
Data structure for motion capture data. Created by the function `mcread`.

**structure**
- **type**: structure type (`'MoCap data'`)  
- **filename**: name of the file from where the data were read  
- **nFrames**: number of frames  
- **nCameras**: number of cameras  
- **nMarkers**: number of markers  
- **freq**: sampling frequency (frame rate) of motion data  
- **nAnalog**: number of analog devices  
- **anaFreq**: sampling frequency of analog data  
- **timerOrder**: order of time differentiation of data (0 = location, 1 = velocity, 2 = acceleration, 3 = jerk/jolt/surge/lurch, 4 = jounce/snap, 5 = crackle, 6 = pop, 7 = you-name-it)  
- **markerName**: marker names (cell structure)  
- **data**: motion capture data (`nFrames x 3nMarkers` matrix)  
- **analogdata**: analog data (`nFrames x nAnalog` matrix)  
- **other**: other data read from the file (depends on equipment and file format used)  
  - for mocap data read from a `.tsv` file, the fields are:  
    - **other.descr**: some description string  
    - **other.timeStamp**: some time stamp string  
    - **other.dataIncluded**: '3D'

**comments**

**see also**
- `mcread`
norm data structure

 synopsis
 Data structure for vector norms. Created by mcnorm.

 structure
 type: structure type ('norm data')
 filename: name of the file from where the data were read
 nFrames: number of frames
 nCameras: number of cameras
 nMarkers: number of markers
 freq: sampling frequency (frame rate) of motion data
 nAnalog: number of analog devices
 anaFreq: sampling frequency of analog data
 timedeOrder: order of time differentiation of data (0 = location, 1 = velocity, 2 = acceleration, 3 = jerk/jolt/surge/lurch, 4 = jounce/snap, 5 = crackle, 6 = pop, 7 = you-name-it)
 markerName: marker names (cell structure)
 data: motion capture data (nFrames x nMarkers matrix)
 analogdata: analog data (nFrames x nAnalog matrix)
 other: other data read from the file (depends on equipment and file format used)

 comments
 The data structure is identical to MoCap data structure, except that the data field contains only one column per marker.

 see also
 mcnorm
**segm data structure**

**synopsis**

Data structure for body segment data. Created by mcj2s.

**structure**

- type: structure type ('segm data')
- filename: name of the file from where the data were read
- nFrames: number of frames
- nCameras: number of cameras
- nMarkers: number of markers
- freq: sampling frequency (frame rate) of motion data
- nAnalog: number of analog devices
- anaFreq: sampling frequency of analog data
- timederOrder: order of time differentiation of data (0 = location, 1 = velocity, 2 = acceleration, 3 = jerk/jolt/surge/lurch, 4 = jounce/snap, 5 = crackle, 6 = pop, 7 = you-name-it)
- analogdata: []
- other: [1x1 struct]
- parent: vector containing the number of the parent joint (proximal joint in kinematic chain) of each joint; zero means no parent joint
- roottrans: matrix (nFrames x 3) containing the coordinates of body root
- rootrot: [1x1 struct]
  - rootrot.az: azimuth angle of the normal vector of the frontal plane (see the j2spar structure)
  - rootrot.el: elevation angle of the normal vector of the frontal plane (see the j2spar structure)
- segm: [1 x nMarkers struct]
  - segm(k).eucl: euclidean vector pointing from proximal to distal joint of the segment
  - segm(k).r: length of the segment (averaged over time)
  - segm(k).quat: quaternion representing rotation from (0 0 -1) to segm(k).eucl
  - segm(k).angle: angle between segments k and k-1
- segmentName: cell structure containing the names of the segments

**comments**

The number of each segment is identical to the number of the marker representing the distal joint of the segment in the MoCap data structure from which the segm data structure was derived.

**see also**

mcj2s
m2jpar parameter structure

**synopsis**
Parameters for conversion from markers to joints.

**structure**
- type: structure type ('m2jpar')
- nMarkers: number of joints
- markerNum: cell structure containing, for each joint, the numbers of the markers whose centroid defines the location of that joint; for instance, if markerNum{k} = [m1 m2 m3 m4], the location of joint k is calculated as the centroid of markers m1, m2, m3, and m4
- markerName: cell structure containing the names of the joints

**comments**

**see also**
- mcinitm2jpar
j2spar parameter structure

**synopsis**
Parameters for conversion from joints to segments.

**structure**
- type: structure type ('j2spar')
- rootMarker: number of the root joint
- frontalPlane: numbers of three joints that define the frontal plane
- parent: vector containing the number of the parent joint (proximal joint in kinematic chain) of each joint; zero means no parent joint
- segmentName: cell structure containing the names of each segment

**comments**
The parent number of the root joint is zero.

**see also**
- mcinitj2spar
animpar parameter structure

**synopsis**
Parameters for creating frame plots and animations.

**structure**
- **type**: structure type ('animpar')
- **scrsize**: two-component vector containing the size of frames in pixels [width height]
- **limits**: four-component vector containing the limits of x and z coordinates for plotting [xmin xmax zmin zmax]
- **az**: azimuth angle (in degrees) of viewing point
- **el**: elevation angle (in degrees) of viewing point
- **msize**: size of markers
- **colors**: five-character string containing the colors of background, markers, connections, traces, and marker numbers, respectively
- **markercolors**: string containing the individual colors for the markers
- **conncolors**: string containing the individual colors for the connector lines
- **tracecolors**: string containing the individual colors for the trace lines
- **numbercolors**: string containing the individual colors for numbers
- **cwidth**: width of connection lines
- **twidth**: width of trace lines
- **conn**: matrix (nMarkers x 2) indicating the connections between markers; each row represents one connection, with the numbers indicating the markers to be connected
- **conn2**: matrix (nMarkers x 4) indicating the connections between midpoints of two marker pairs; each row represents one connection, with the first two numbers and the last two numbers indicating the markers whose midpoints are to be connected
- **trm**: vector indicating the markers with a trace
- **trl**: length of trace in seconds
- **showmnum**: flag indicating whether marker numbers are shown (1=yes, 0=no)
- **numbers**: array indicating the markers for which number is to be shown
- **showfnum**: flag indicating whether frame numbers are shown (1=yes, 0=no)
- **animation**: flag indicating whether animation is created (1=yes, 0=no); this is set by the mcanimate function before it calls the mcplotframe function
- **fps**: frames per second used in animation
- **folder**: name of the folder created for saving animation frames
- **pers**: structure for perspective projection parameters:
  - **pers.c**: 3D position of the camera
  - **pers.th**: orientation of the camera
  - **pers.e**: viewer's position relative to the display surface

**comments**

**see also**
- mcinitanimpar
Function Reference
mc2frontal

**synopsis**
Rotates motion-capture data to have a frontal view with respect to a pair of markers.

**syntax**
```matlab
d2 = mc2frontal(d, m1, m2);
d2 = mc2frontal(d, m1, m2, method);
```

**input parameters**
- `d`: MoCap data structure or data matrix
- `m1, m2`: numbers of the markers that define the frontal plane
- `method`: rotation method, possible values:
  - 'mean' (default) rotates data in all frames with the same angle to have a frontal view with respect to the mean locations of markers `m1` and `m2`
  - 'frame' rotates each frame separately to have a frontal view with respect to the instantaneous locations of markers `m1` and `m2`; with this method, each individual frame is centered as well

**output**
- `d2`: MoCap data structure or data matrix

**examples**
```matlab
d2 = mc2frontal(d, 3, 7);
d2 = mc2frontal(d, 3, 7, 'frame');
```

**comments**
The frontal plane is defined by the temporal mean of markers `m1` and `m2`

`mc2frontal(d, 1, 2)` would rotate to that view:

**see also**
- `mcrotate`
mcanimate

**synopsis**
Creates png frames for an animation of motion capture data, and saves them to a folder.

**syntax**
```
par = mcanimate(d);
par = mcanimate(d, par);
par = mcanimate(d, par, proj);
```

**input parameters**
- `d`: MoCap data structure
- `par`: animpar structure
- `proj`: projection used: 0 = orthographic (default), 1 = perspective

**output**
- `par`: animpar structure used for plotting the frames

**examples**
```
mcanimate(d, par, 1);
```

**comments**
If the animpar structure is not given as input argument, the function creates it by calling the function `mcinitanimpar` and setting the `.limits` field of the animpar structure automatically so that all the markers fit into all frames.
If the `par.pers` field (perspective projection) is not given, it is created internally for backwards compatibility. For explanation of the `par.pers` field, see `help mcinitanimpar`

**see also**
- `mcplotframe`, `mcinitanimpar`
mcbandpass

**synopsis**
Band pass filters data in a MoCap or norm structure using an FFT filter.

**syntax**
\[ d2 = mcbandpass(d, f1, f2); \]
\[ d2 = mcbandpass(d, f1, f2, method); \]

**input parameters**
- `d`: MoCap or norm data structure
- `f1`: lower frequency in Hz of passband
- `f2`: higher frequency in Hz of passband
- `method`: filtering window, ‘rect’ (default) or ‘gauss’

**output**
- `d2`: MoCap or norm data structure containing band pass filtered data

**examples**
\[ d2 = mcbandpass(d, 0.5, 3); \]

**comments**

**see also**
**mcboundrect**

**synopsis**
Calculates the bounding rectangle (the smallest rectangular area that contains the projection of the trajectory of each marker on the horizontal plane (i.e., floor).

**syntax**
- `br = mcboundrect(d);`
- `br = mcboundrect(d, mnum);`
- `br = mcboundrect(d, mnum, w, hop);`

**input parameters**
- `d`: MoCap data structure
- `mnum`: marker numbers
- `w`: length of analysis window (default: 4 sec)
- `hop`: overlap of analysis windows (default: 2 sec)

**output**
- `br`: data matrix (windows x nMarkers)

**examples**
- `br = mcboundrect(d);`
- `br = mcboundrect(d, [1 3 5]);`
- `br = mcboundrect(d, [1:d.nMarkers], 3, 1);`

**comments**
If the function is called with the mocap data structure as the only input parameter, the calculation is performed for all markers with the default parameters. If the window and overlap length are to be changed, the markers have to be always specified (e.g., all markers by `[1:d.nMarkers]`).

**see also**
mccenter

**synopsis**
Translates motion capture data to have a centroid of \([0 0 0]\) across markers and over time.

**syntax**
d2 = mccenter(d);

**input parameters**
d: MoCap data structure or data matrix

**output**
d2: MoCap data structure or data matrix

**examples**

**comments**
Missing data (NaN's) is ignored when calculating the centroid.

**see also**
**mccreateconnmatrix**

**synopsis**
Creates a connection matrix for the animation parameters (.conn field) by using the "bones" connections saved as a label list of the Qualisys track manager software (QTM).

**syntax**
```
par = mccreateconnmatrix(fn, par);
```

**input parameters**
- *fn*: text file (ending: .txt) that contains the "bones" (connections) made in QTM
- *par*: animpar structure

**output**
- *par*: animpar structure with connection matrix

**examples**
```
par = mccreateconnmatrix('labellist.txt', par);
```

**comments**
This function works only with label list files created by Qualisys Track Manager. This function works for marker representations (before any marker reduction or joint transformation has been applied). The markers in the MoCap structure must resemble the structure of the marker connections in the label list file.

**see also**
- mcinitanimpar


mccumdist

**Synopsis**
Calculates the cumulative distance traveled by each marker.

**Syntax**
d2 = mccumdist(d);

**Input Parameters**
d: MoCap data or norm data structure

**Output**
d2: norm data structure

**Examples**

**Comments**
If the input consists of one-dimensional data (i.e., norm data), the cumulative distance to the origin of the reference space/coordination system is calculated, which is not (necessarily) the cumulated distance traveled by the marker.

**See also**
mccut

synopsis
Cuts two MoCap structures to the length of the shorter one.

syntax
\[ [d_1, d_2] = \text{mccut}(d_1, d_2); \]

input parameters
d1, d2: MoCap or norm structures

output
d1, d2: MoCap or norm structures, one shortened and one original (both with same number of frames)

examples

comments

see also

MoCap Toolbox Manual
mcdecompose

**synopsis**
Decomposes a kinematic variable into tangential and normal components.

**syntax**
```
[dt, dn] = mcdecompose(d, order);
```

**input parameters**
d: MoCap data structure containing either location or velocity data (timederorder = 0 or 1)
order: time derivative order of the variable, must be at least 2 (2 = acceleration, 3 = jerk, etc.)

**output**
dt: norm data structure containing the tangential components
dn: norm data structure containing the normal components

**examples**
```
[dt, dn] = mcdecompose(d, 2); % acceleration
[dt, dn] = mcdecompose(d, 3); % jerk
[dt, dn] = mcdecompose(d, 4); % jounce / snap
[dt, dn] = mcdecompose(d, 5); % crackle
[dt, dn] = mcdecompose(d, 6); % pop
[dt, dn] = mcdecompose(d, 7); % you-name-it
```

**comments**

**see also**

**references**
mceigenmovement

**synopsis**
Constructs eigenmovements using PCA and a scaled sinusoidal projection.

**syntax**
```
e = mceigenmovement(d);
e = mceigenmovement(d, eigind);
e = mceigenmovement(d, eigind, len);
e = mceigenmovement(d, eigind, len, per);
```

**input parameters**
- `d`: MoCap or segm data structure
- `eigind` (optional): selected eigenmovements (if not given, projections onto the first PCs that contain a total of 90% of the variance are returned)
- `len` (optional): length in seconds (default 0.5 sec)
- `per` (optional): period in seconds (default 0.5 sec)

**output**
e: vector of MoCap or segm data structures

**examples**
```
e = mceigenmovement(d);
e = mceigenmovement(d, 1:3);
e = mceigenmovement(d, 1:4, 2);
e = mceigenmovement(d, 1:2, 1.2, 0.6);
```

**comments**
The sinusoidal projections are scaled to match the RMS amplitudes of the PC projections of respective degrees of freedom.

**see also**
- mcpcaproj
mcfillgaps

**synopsis**
Fills gaps in motion capture data.

**syntax**

```matlab
d2 = mcfillgaps(d);
d2 = mcfillgaps(d, maxfill);
d2 = mcfillgaps(d, method);
d2 = mcfillgaps(d, maxfill, method);
```

**input parameters**
- **d**: MoCap, norm, or segm data structure
- **maxfill**: maximal length of gap to be filled in frames (optional, default = 1000000)
- **method**: three different options for filling missing frames in the beginning and/or end of a recording:
  - default (parameter empty): missing frames in the beginning and/or in the end are set to 0;
  - 'fillall': fills missing frames in the beginning and end of the data with the first actual (recorded) value or the last actual (recorded) value respectively;
  - 'nobefill': fills all the gap in the data, but not missing frames in the beginning or end of the data, but sets them to NaN instead.

**output**
- **d2**: MoCap, norm, or segment data structure

**examples**

```matlab
d2 = mcfillgaps(d);
d2 = mcfillgaps(d, 120);
d2 = mcfillgaps(d, 'nobefill');
d2 = mcfillgaps(d, 60, 'fillall');
```

**comments**
Uses linear interpolation. More sophisticated algorithms will be implemented in the future.

**see also**
mcfilteremg

**synopsis**
Filters EMG data.

**syntax**
```
out = mcfilteremg(emgdata);
out = mcfilteremg(emgdata, filterfreqs);
```

**input parameters**
- `emgdata`: norm data structure containing EMG data
- `filterfreqs`: cutoff frequencies (in Hz) for the Butterworth filters; first value for high-pass filter, second value for low-pass filter (default: [20 24])

**output**
- `out`: norm data structure containing filtered data

**examples**
```
out = mcfilteremg(emgdata);
out = mcfilteremg(emgdata, [18 21]);
```

**comments**
Filters the data using a 4th order Butterworth high-pass filter (default cutoff frequency: 20 Hz), then full-wave rectifies it, then filters it using a 4th order Butterworth low-pass filter (default cutoff frequency: 24 Hz).

**see also**
- mcreademg
mcgetmarker

synopsis
Extracts a subset of markers.

syntax
\[ d2 = \text{mcgetmarker}(d, \text{mnum}); \]

input parameters
d: MoCap or norm data structure
mnum: vector containing the numbers of markers to be extracted

output
d2: MoCap or norm data structure

examples
\[ d2 = \text{mcgetmarker}(d, [1 3 5]); \]

comments

see also

MoCap Toolbox Manual
mcgetmarkernames

**synopsis**
Returns the names of markers.

**syntax**

```matlab
mn = mcgetmarkernames(d);
```

**input parameters**

- `d`: MoCap data structure

**output**

- `mn`: cell structure containing marker names

**examples**

**comments**

**see also**

MoCap Toolbox Manual
mcgetsegmpar

**synopsis**
Get parameters for body segments.

**syntax**
```matlab
spar = mcgetsegmpar(model, segmnum);
```

**input parameters**
- model: string indicating the body-segment model used (possible value: 'Dempster', more to be added in the future)
- segmnum: vector indicating numbers for each segment

**output**
- spar: segmpar structure

**examples**
```matlab
segmnum = [0 0 8 7 6 0 8 7 6 13 12 10 11 3 2 1 11 3 2 1];
spar = mcgetsegmpar('Dempster', segmnum);
```

**comments**
Returns the mass relative to total body mass (`spar.m`), relative distance of center of mass from proximal joint (`spar.comprox`) and distal joint (`spar.comdist`), and radius of gyration relative to center of gravity (`spar.rogcg`), proximal joint (`spar.rogprox`) and distal joint (`spar.rogdist`) of for body segments indicated in `segmnum` according to given body-segment model.

Segment number values for model 'Dempster': no parameter=0, hand=1, forearm=2, upper arm=3, forearm and hand=4, upper extremity=5, foot=6, leg=7, thigh=8, lower extremity=9, head=10, shoulder=11, thorax=12, abdomen=13, pelvis=14, thorax and abdomen=15, abdomen and pelvis=16, trunk=17, head, arms and trunk (to glenohumeral joint)=18, head, arms and trunk (to mid-rib)=19.

Note that the root needs its own segment being 0, so `segmnum` is of size segments+1. (The first zero in the `segmnum` vector above).

See the description of the segmpar structure.

**see also**

**references**
mchilbert

**synopsis**
Calculates the Hilbert transform of data in a MoCap or norm structure.

**syntax**
```
[amp, phase, h] = mchilbert(d);
```

**input parameters**
d: MoCap or norm data structure

**output**
amp: amplitude of analytic function derived from zero-mean signal
phase: (unwrapped) phase of analytic function derived from zero-mean signal
h: analytic function

**examples**

**comments**
See help hilbert

**see also**
mchilberthuang

synopsis
Performs a Hilbert-Huang transform of order N on MoCap, norm or segm data.

syntax
hh = mchilberthuang(d, N);

input parameters
d: MoCap, norm or segm data structure
N: order of the H-H transform

output
hh: vector of MoCap, norm or segm data structures containing H-H transforms

examples

comments
See help hilberthuang

see also
mcicaproj

**synopsis**
Performs an Independent Components analysis on MoCap, norm or segm data, using the FastICA algorithm, and projects the data onto selected components.

**syntax**
```
[di, p] = mcicaproj(d, pc, ic);
```

**input parameters**
- d: MoCap, norm or segm data structure
- pc: number of PCs entered into ICA
- ic: number of ICs estimated

**output**
- di: vector of MoCap, norm or segm data structures
- p: structure containing the following fields:
  - icasig: independent components
  - A: mixing matrix
  - W: separation matrix
  - meanx: mean vector of variables

**examples**
```
[di, p] = mcicaproj(d, 6, 3);
```

**comments**

**see also**
mcpcaproj, mcsethares
mcinitanimpar

**synopsis**

Initializes an animation parameter (animpar) structure.

**syntax**

```c
ap = mcinitanimpar;
```

**input parameters**

(none)

**output**

- `ap`: animation parameter (animpar) structure

**examples**

**comments**

See also description of the animpar structure (initialized values given in parentheses:)

- `scrsizesize`: frame size in pixels ([800 600])
- `limits`: plot limits [xmin xmax zmin zmax] ([])  
- `az`: azimuth vector in degrees (0)
- `el`: elevation vector in degrees (0)
- `msize`: marker size (12)
- `colors`: [background marker connection trace markernumber] ('kwwww')
- `markercolors`: String holding marker colors ("")
- `conncolors`: String holding connector (line) colors ("")
- `tracecolors`: String holding trace colors (only animations) ("")
- `numbercolors`: String holding number colors (indicated in the numbers array) ("")
- `cwidth`: width of connectors (either single value or vector with entries for different widths) (1)
- `twidth`: width of traces (either single value or vector with entries for different widths) (1)
- `conn`: marker-to-marker connectivity matrix (M x 2) - mccreateconnmatrix() can be used for creating the connection matrix ([])
- `conn2`: midpoint-to-midpoint connectivity matrix (M x 4) ([])
- `trm`: vector indicating markers for which traces are added ([])
- `trl`: length of traces in seconds (0)
- `showmnum`: show marker numbers, 1=yes, 0=no (0)
- `numbers`: array indicating the markers for which number is to be shown ([])
- `showfnum`: show frame numbers, 1=yes, 0=no (0)
- `animation`: save jpeg frames to folder, 1=yes, 0=no (0)
- `fps`: frames per second for animation (30)
- `folder`: folder name for output animation in jpeg frames ('tmp')

pers: perspective projection parameters:

- `pers.c`: 3D position of the camera [0 -4000 0]
pers.th: orientation of the camera [0 0 0]
pers.e: viewer's position relative to the display surface [0 -2000 0]

see also
mccreateconnexion, mcplotframe, mcanimate
**mcinitj2spar**

**synopsis**
Initialises the parameter structure for joint-to-segment mapping.

**syntax**

\[
\text{par} = \text{mcinitj2spar};
\]

**input parameters**

(none)

**output**

\[
\text{par}: \text{j2spar structure}
\]

**examples**

**comments**
See explanation about the j2spar structure. The initialized values are as follows:

- **type**: 'j2spar'
- **rootMarker**: 0
- **frontalPlane**: [1 2 3]
- **parent**: []
- **segmentName**: {}

The fields `par.parent` and `par.segmentName` have to be entered manually.

**see also**

- `mcj2s`
mcinitm2jpar

**synopsis**
Initialises the parameter structure for marker-to-joint mapping.

**syntax**

```
par = mcinitm2jpar;
```

**input parameters**

(none)

**output**

par: m2jpar structure

**examples**

**comments**

See the explanation of the m2jpar structure. The initialized values are as follows:

- type: 'm2jpar'
- nMarkers: 0
- markerNum: {}
- markerName: {}

The fields `par.nMarkers`, `par.markerNum` and `par.markerName` have to be entered manually.

**see also**

mcm2j
mcinitstruct

**synopsis**
Initializes MoCap or norm data structure.

**syntax**
```
d1 = mcinitmocap;
d1 = mcinitmocap(type);
d1 = mcinitmocap(type, data, freq);
d1 = mcinitmocap(type, data, freq, markerName);
d1 = mcinitmocap(type, data, freq, markerName, fn);
d1 = mcinitmocap(data, freq);
d1 = mcinitmocap(data, freq, markerName);
d1 = mcinitmocap(data, freq, markerName, fn);
```

**input parameters**
- **type**: 'MoCap data' or 'norm data' (default: 'MoCap data')
- **data**: data to be used in the .data field of the mocap structure (default: [])
- **freq**: frequency / capture rate of recording (default: NaN)
- **markerName**: cell array with marker names (default: {})
- **fn**: filename (default: '')

**output**
- **d1**: mocap or norm data structure with default parameters or parameter adjustment according to the parameter input.

**examples**
```
d1 = mcinitmocap;
d1 = mcinitmocap('norm data', data);
d1 = mcinitmocap(data, 120, markernames, 'mydata1.tsv');
```

**comments**
default parameters (for 'MoCap data '):
```
type: 'MoCap data'
filename: ''
nFrames: 0
nCameras: NaN
nMarkers: 0
freq: NaN
nAnalog: 0
anaFreq: 0
timederOrder: 0
markerName: {}
data: []
```
analogdata: []
other:
    other.descr: 'DESCRIPTION --'
    other.timeStamp: 'TIME_STAMP--'
    other.dataIncluded: '3D'

see also
**mcj2s**

**synopsis**
Performs a joint-to-segment mapping.

**syntax**
\[ d2 = mcj2s(d, \text{par}); \]

**input parameters**
- \( d \): MoCap data structure
- \( \text{par} \): j2spar structure

**output**
- \( d2 \): segm data structure

**examples**

**comments**
See explanation of the j2spar structure.

**see also**
- mcinitj2spar, mcs2j
mckinenergy

 synopsis
 Estimates the instantaneous kinetic energy of each body segment.

 syntax
 [te, re] = mckinenergy(d, segd, spar);

 input parameters
 d: MoCap data structure
 segd: segm data structure calculated from d
 spar: segmpar structure (see mcgetsegmpar)

 output
 te: matrix containing translational energy values for each body segment
 re: matrix containing rotational energy values for each body segment

 examples
 segd = mcj2s(d, j2spar);
 spar = mcgetsegmpar('Dempster', segmnum);
 [te, re] = mckinenergy(d, segd, spar);

 comments
 The energy for a given segment is in the column corresponding to the number of the distal joint of the respective segment.

 see also
 mcj2s, mcgetsegmpar, mcpotenergy
mckurtosis

**synopsis**
Calculates the kurtosis of data, ignoring missing values.

**syntax**
\[
m = mckurtosis(d);
\]

**input parameters**
d: MoCap data structure, norm data structure, or data matrix.

**output**
m: row vector containing the kurtosis values of each data column

**examples**

**comments**

**see also**
mcmean, mcstd, mcvar, mcskewness
mcm2j

**synopsis**
Performs a marker-to-joint mapping.

**syntax**
```
d2 = mcm2j(d, par);
```

**input parameters**
- `d`: MoCap data structure
- `par`: m2jpar structure

**output**
- `d2`: MoCap data structure

**examples**

**comments**
The fields `par.nMarkers`, `par.markerNum` and `par.markerName` have to be entered manually.
See the explanation of the m2jpar structure.

**see also**
- mcinitm2jpar
mcmarkerdist

**synopsis**
Calculates the frame-by-frame distance of a marker pair.

**syntax**

```matlab
dist = mcmarkerdist(d, m1, m2);
```

**input parameters**
- d: MoCap data structure
- m1, m2: marker numbers

**output**
- dist: column vector

**examples**

```matlab
dist = mcmarkerdist(d, 1, 5);
```

**comments**

**see also**

MoCap Toolbox Manual
mcmean

**synopsis**
Calculates the temporal mean of data, ignoring missing values.

**syntax**

```matlab
m = mcmean(d);
```

**input parameters**

- `d`: MoCap data structure, norm data structure, or data matrix.

**output**

- `m`: row vector containing the means of each data column

**examples**

**comments**

**see also**

- `mcstd`, `mcvar`, `mcskewness`, `mckurtosis`
mcmerge

**synopsis**
Merges two MoCap data structures and optionally the corresponding animation parameter files.

**syntax**
```
d3 = mcmerge(d1, d2);
[d3, p3] = mcmerge(d1, d2, p1, p2);
```

**input parameters**
- d1, d2: MoCap data structures
- p1, p2: animpar structures for d1 and d2

**output**
- d3: MoCap data structure
- p3: animpar structure

**examples**

**comments**
D1 and d2 must have identical frame rates. If the numbers of frames are not equal, the MoCap data structure with the higher number of frames will be cut before merging.

**see also**
mcmissing

**Synopsis**
Reports missing data per marker and frame.

**Syntax**

```
[mf, mm, mgrid] = mcmissing(d);
```

**Input Parameters**

d: MoCap or norm data structure.

**Output**

mf: number of missing frames per marker
mm: number of missing markers per frame
mgrid: matrix showing missing data per marker and frame (rows correspond to frames and columns to markers)

**Examples**

**Comments**

**See Also**
mcnorm

**synopsis**
Calculates the norms of kinematic vectors.

**syntax**
```matlab
n = mcnorm(d);
n = mcnorm(d, comps);
```

**input parameters**
- `d`: MoCap data structure
- `comps`: components included in the calculation (optional, default = 1:3)

**output**
- `n`: norm data structure

**examples**
```matlab
n = mcnorm(d);
n = mcnorm(d, 1:2); % calculates norm of horizontal projection
```

**comments**

**see also**
mcpcaproj

synopsis
Performs a Principal Components analysis on MoCap, norm or segm data and projects the data onto selected components.

syntax
[dp, p] = mcpcaproj(d);
[dp, p] = mcpcaproj(d, pc);
[dp, p] = mcpcaproj(d, pc, proj);

input parameters
d: MoCap, norm or segm data structure
pc (optional): selected Principal Components (if not given, projections onto the first PCs that contain a total of 90% of the variance are returned)
proj (optional): projection function (if not given, the PC projections of the data in d are used)

output
dp: vector of MoCap, norm or segm data structures
p: structure containing the following fields:
l: proportion of variance contained in each PC
q: PC vectors (columns)
c: PC projections (rows)
meanx: mean vector of variables

examples
[dp, p] = mcpcaproj(d);
[dp, p] = mcpcaproj(d, 1:3);
[dp, p] = mcpcaproj(d, 1:3, sin(2*pi*0:60/60));

comments

see also
mcicaproj, mcsethares
mcperiod

synopsis
Estimates the period of movement for each marker and each dimension.

syntax
[per, ac, eac, lag] = mcperiod(d);
[per, ac, eac, lag] = mcperiod(d, maxper);
[per, ac, eac, lag] = mcperiod(d, method);
[per, ac, eac, lag] = mcperiod(d, maxper, method);

input parameters
d: MoCap or norm data structure
maxper: maximal period in seconds (optional, default = 2 secs)
method: sets if 'first' or 'highest' maximal value of the autocorrelation function is taken as periodicity estimation (optional, default: 'first')

output
per: row vector containing period estimates for each column
ac: matrix containing autocorrelation functions for each column
eac: matrix containing enhanced autocorrelation functions for each column
lag: vector containing lag values for the (normal and enhanced) autocorrelation functions

examples
[per, ac, eac, lag] = mcperiod(d, 3);
per = mcperiod(d, 'highest');

comments
In ac and eac, each column corresponds to a dimension of a marker (or in case of norm data to a marker), and each row corresponds to a time lag.

see also

references
mcplotframe

synopsis
Plots frames of motion capture data.

syntax
par = mcplotframe(d, n);
par = mcplotframe(d, n, par);
par = mcplotframe(d, n, par, proj);

input parameters
d: MoCap data structure
n: vector containing the numbers of the frames to be plotted
par: animpar structure
proj: projection used: 0 = orthographic (default), 1 = perspective

output
par: animpar structure

examples
par = mcplotframe(d, 1);
mcplotframe(d, 500:10:600, par, 1);

comments
If the animpar structure is not given as input argument, the function creates it by calling the function mcinitanimpar and setting the .limits field of the animpar structure automatically so that all the markers fit into all frames.
If the par.pers field (perspective projection) is not given, it is created internally for backwards compatibility. For explanation of the par.pers field, see help mcinitanimpar.

see also
mcanimate, mcinitanimpar
mcplotphaseplane

**synopsis**
Plots motion capture data on a phase plane.

**syntax**
- `mcplotphaseplane(d1, d2, marker, dim)` % for MoCap data structure
- `mcplotphaseplane(n1, n2, marker)` % for norm data structure
- `mcplotphaseplane(s1, s2, segm, var)` % for segm data structure

**input parameters**
- `d1, d2, n1, n2, s1, s2`: MoCap data structure, norm data structure, or segm data structure
- `marker`: vector containing marker numbers to be plotted (for MoCap and norm data structure)
- `dim`: vector containing dimensions to be plotted (for MoCap data structure)
- `segm`: body segment number (for segm data structure)
- `var`: variable to be plotted for segment `segm` (for segm data structure)

**output**
Figure.

**examples**
- `mcplotphaseplane(d1, d2, 1:3, 3)` % for MoCap data structure
- `mcplotphaseplane(n1, n2, 5)` % for norm data structure
- `mcplotphaseplane(s1, s2, [2 6 20], 'angle')` % for segm data structure
- `mcplotphaseplane(s1, s2, 5:10, 'eucl')` % for segm data structure
- `mcplotphaseplane(s1, s2, [12 14], 'quat')` % for segm data structure

**comments**

**see also**
mcplottimeseries

dynopsis
Plots motion capture data as time series.

syntax
mcplottimeseries(d, marker) % for MoCap or norm data structure
mcplottimeseries(d, marker, dim) % specifying dimensions to be plotted
mcplottimeseries(d, marker, timetype) % sec or frames as axis unit
mcplottimeseries(d, marker, plotoption) % combined or separate plots
mcplottimeseries(d, marker, dim, timetype)
mcplottimeseries(d, marker, dim, plotoption)
mcplottimeseries(d, marker, dim, timetype, plotoption)
mcplottimeseries(s, segm, var) % for segm data structure
mcplottimeseries(s, segm, var, timetype)

input parameters
d/s: MoCap data structure, norm data structure, or segm data structure
marker: vector containing marker numbers to be plotted (for MoCap and norm data structure)
segm: body segment number (for segm data structure)
dim: vector containing dimensions to be plotted (for MoCap data structure)
var: variable to be plotted for segment segm (for segm data structure)
timetype: time type used in the plot (seconds (default) or 'frame')
plotoption: plotting option (for MoCap or norm data structure); separate (default) or 'comb':
  separate: all time series are plotted in separate subplots
  comb: all time series will be plotted into the same plot using different colors

output
Figure.

examples
mcplottimeseries(d, 2) % MoCap or norm data structure, marker 2, first dim
mcplottimeseries(d, 1:3, 3) % markers 1 to 3, third dim
mcplottimeseries(d, 1:3, 3, 'comb') % same plot, different colors per dim
mcplottimeseries(d, 5, 'frame') % frames as x axis unit
mcplottimeseries(d, 5, 'frame', 'comb')
mcplottimeseries(s, [3 6 20], 'angle') % for segm data structure
mcplottimeseries(s, 5:10, 'eucl', 'frame') % frames as x axis unit
mcplottimeseries(s, [12 14], 'quat')
comments
   For segment data, the plot option 'comb' is not implemented yet.

see also
mcpotenergy

synopsis
Estimates the instantaneous potential energy of each body segment.

syntax
pe = mcpotenergy(d, segd, segmpar)

input parameters
- d: MoCap data structure
- segd: segm data structure calculated from d
- segmpar: segmpar structure (see mcgetsegmpar)

output
pe = matrix containing potential energy values for each body segment

examples
segd = mcj2s(d, j2spar);
spar = mcgetsegmpar('Dempster', segmnum);
pe = mckinenergy(d, segd, spar);

comments
The energy for a given segment is in the column corresponding to the number of the distal joint of the respective segment.

see also
mcj2s, mcgetsegmpar, mckinenergy
mcread

synopsis
Reads a motion capture data file and returns a MoCap data structure.

syntax
\[ d = \text{mcread}(\text{fn}); \]
\[ d = \text{mcread}; \]

input parameters
fn: file name, tsv, c3d, mat, or wii format. If no input parameter is given, a file open dialog opens.

output
d: MoCap data structure containing parameter values and data

examples
\[ d = \text{mcread}('filename.tsv'); \]
\[ d = \text{mcread}('filename.c3d'); \]
\[ d = \text{mcread}('filename.mat'); \]
\[ d = \text{mcread}('filename.wii'); \]
\[ d = \text{mcread}; \]

comments
Currently the .c3d, .tsv (as exported by QTM), .mat (as exported by QTM), and .wii (WiiDataCapture software) formats are supported. The file names must have postfixes '.c3d', '.tsv', '.mat', or '.wii', respectively. For reading .c3d files, the function provided at http://www.c3d.org/download_apps.html is used.
For exporting in .tsv format from Qualisys QTM, recommended export parameter are:
3D data and Include TSV header ticked
Export time data for every frame and write column headers will be ignored by mcread if ticked
Note: The .c3d format does not support more than 65535 frames per file (see www.c3d.org/HTML/default.htm ➔ The C3D file format ➔ Limitations). Therefore, if you happen to have longer recordings, export them either in .tsv or .mat, or in more than one c3d file.

see also

MoCap Toolbox Manual 106
mcreademg

**synopsis**
Reads emg files in .tsv format recorded with the Mega EMG system using QTM.

**syntax**
\[ d = \text{mcreademg}(\text{fn}); \]

**input parameters**
fn: File name; tsv format (norm data structure)

**output**
d: norm data structure

**examples**
\[ d = \text{mcreademg('filename.tsv')}; \]

**comments**

**see also**
mcfilteremg
mcreorderdims

**synopsis**
Reorders the Euclidean dimensions in motion capture data.

**syntax**
```matlab
d2 = mcreorderdims(d, dims);
```

**input parameters**
- d: MoCap data structure
- dims: vector containing the new order of dimensions

**output**
- d2: MoCap data structure

**examples**
```matlab
d2 = mcreorderdims(d, [1 3 2]);
```

**comments**

**see also**
mcresample

synopsis
Resamples motion capture data using interpolation.

syntax
\[ d2 = \text{mcresample}(d, \text{newfreq}, \text{method}); \]

input parameters
- \( d \): MoCap data structure
- \( \text{newfreq} \): new frame rate
- \( \text{method} \): interpolation method (optional, default 'linear'; for other options, see \text{help interp1})

output
- \( d2 \): MoCap data structure

examples
\[ d2 = \text{mcresample}(d, 240); \]
\[ d2 = \text{mcresample}(d, 360, 'spline'); \]

comments

see also

MoCap Toolbox Manual
**mcrotate**

**synopsis**
Rotates motion-capture data.

**syntax**

```matlab
d2 = mcrotate(d, theta);
d2 = mcrotate(d, theta, axis);
d2 = mcrotate(d, theta, point);
d2 = mcrotate(d, theta, axis, point);
```

**input parameters**
- `d`: MoCap data structure or data matrix
- `theta`: rotation angle (in degrees)
- `axis`: rotation axis (optional, default = `[0 0 1]`)
- `point`: point through which the rotation axis goes (optional, default is the centroid of markers over time)

**output**
- `d2`: MoCap data structure or data matrix

**examples**

```matlab
d2 = mcrotate(d, 130); % rotate 130 degrees counterclockwise around the vertical axis
d2 = mcrotate(d, 90, [1 0 0]); % rotate around the x axis
d2 = mcrotate(d, 45, [0 1 0], [0 0 500]); % rotate around the axis parallel to y axis going through point [0 0 500]
d2 = mcrotate(d, 20, [], [0 1000 0]); % rotate around the z (vertical) axis going through point [0 1000 0]
```

**comments**
If theta is a vector, its values are used as evenly-spaced break points in interpolation. This allows the creation of dynamic rotation of the data.

Rotation is performed according to the right-hand rule. For instance, if the rotation axis is pointing vertically upwards, positive rotation angle means counterclockwise rotation when viewed from up.

**see also**
- `mc2frontal`
mcs2j

**synopsis**
Perform a segment-to-joint mapping.

**syntax**
\[ d2 = mcj2s(d, par); \]

**input parameters**
- `d`: segm data structure
- `par`: j2spar structure

**output**
- `d2`: MoCap data structure

**examples**

**comments**
See the description of the j2spar structure.

**see also**
- `mcinitj2spar`, `mcj2s`
mcs2posture

 synopsis
 Creates a posture representation from segm data by setting root transition and root rotation to zero values.

 syntax
 p = mcs2posture(d);

 input parameters
 d: segm data structure

 output
 p: segm data structure

 examples

 comments

 see also
 mcj2s
mcsegmangle

**synopsis**
Calculates the angles between two markers.

**syntax**
```
dn = mcsegmangle(d, m1, m2);
```

**input parameters**
- d: MoCap data structure
- m1: marker one
- m2: marker two

**output**
- dn: norm data structure containing the three angles

**examples**
```
dn = mcsegmangle(d, 1, 2);
```

**comments**

**see also**
mcsethares

**synopsis**
Performs either an m-best or a small-to-large Sethares transform on MoCap, norm or segm data.
Returns the basis functions for each DOF for given periods and, with the m-best transform, also the powers for the respective periods.

**syntax**
```
ds = mcsethares(d, per); % small-to-large Sethares transform
[ds, pers, pows] = mcsethares(d, per, nbasis); % m-best Sethares transform
```

**input parameters**
d: MoCap, norm or segm data structure
per: period in frames in case of small-to-large Sethares transform
maximum period in frames in case of m-best Sethares transform
nbasis: number of basis functions estimated (only for m-best Sethares transform)

**output**
ds: MoCap, norm or segm data structure - the only output in case of small-to-large Sethares transform
in case of m-best Sethares transform also:
per: best periods for each degree of freedom
pow: powers of respective periods

**examples**

**comments**
Dependent on the given input parameter, either the m-best or the small-to-large Sethares transform is chosen. See syntax above about in- and output argument structure.
Uses the Periodicity Toolbox downloadable at
[http://eceserv0.ece.wisc.edu/~sethares/downloadper.html](http://eceserv0.ece.wisc.edu/~sethares/downloadper.html)

**see also**
mcpcaproj, mcicaproj
mcsetmarker

**synopsis**
Replaces a subset of markers in an existing mocap or norm structure.

**syntax**
\[
d2 = mcsetmarker(d_orig, d_repl, mnum);
\]

**input parameters**
- `d_orig`: MoCap or norm data structure (the one to be changed)
- `d_repl`: MoCap or norm data structure (the one that contains the replacement data)
- `mnum`: vector containing the marker numbers to be replaced (in the order of the markers in the replacement mocap structure)

**output**
- `d2`: MoCap structure

**examples**
\[
d2 = mcsetmarker(d, d1, [1 3 5]);
\]

**comments**
If the resulting mocap structure shall contain more markers than the original, the data will be appended at the specified marker number. Possible in-between markers will be set to NaN. Marker names will be set to EMPTY and can be adapted manually if desired.

**see also**
mcsimmat

**synopsis**
Calculates self-similarity matrix from MoCap or segm data.

**syntax**
```matlab
sm = mcsimmat(d);
sm = mcsimmat(d, metric);
```

**input parameters**
- `d`: MoCap or segm data structure
- `metric`: distance metric used, see `help pdist` (default: cityblock)

**output**
- `sm`: self-similarity matrix

**examples**
```matlab
sm = mcsimmat(d);
sm = mcsimmat(d, 'corr');
```

**comments**

**see also**
mcskewness

synopsis
Calculates the skewness of data, ignoring missing values.

syntax
m = mcskewness(d);

input parameters
d: MoCap data structure, norm data structure, or data matrix

output
m: row vector containing the skewness values of each data column

examples

comments

see also
mcmean, mcstd, mcvar, mckurtosis
mcsmoothen

synopsis
Smoothens motion capture data using a Butterworth (fast) or a Savitzky-Golay FIR (accurate) smoothing filter.

syntax
\[ d2 = \text{mcsmoothen}(d); \]
\[ d2 = \text{mcsmoothen}(d, \text{filterparams}); \]
\[ d2 = \text{mcsmoothen}(d, \text{method}); \]
\[ d2 = \text{mcsmoothen}(d, \text{window}); \]

input parameters
\- d: MoCap data structure or segm data structure
\- filterparams: order and cutoff frequency for Butterworth filter (optional, default [2, 0.2])
\- method: Butterworth filtering is default - if Savitzky-Golay filtering is to be used, use 'acc' as method argument
\- window: window length (optional, default = 7) for Savitzky-Golay FIR smoothing filter
  (if input is scalar or a string, Savitzky-Golay filter is chosen - if input is vector, it is considered as parameters for Butterworth filter)

output
\- d2: MoCap data structure or segm data structure

examples
\[ d2 = \text{mcsmoothen}(d); \text{ % Butterworth filter smoothing with default parameters} \]
\[ d2 = \text{mcsmoothen}(d, [2, .1]); \text{ % second order Butterworth filter with 0.1 Hz} \]
\text{cutoff frequency}
\[ d2 = \text{mcsmoothen}(d, \text{'acc'}); \text{ % S-G filter smoothing with default frame length} \]
\[ d2 = \text{mcsmoothen}(d, 9); \text{ % S-G filter smoothing using a 9-frame window} \]

comments
The default parameters for the Butterworth filter create a second-order zero-phase digital Butterworth filter with a cutoff frequency of 0.2 Hz.
For information about the Savitzky-Golay filter, see help sgolayfilt.

see also
mctimededer
mcspectrum

**synopsis**
Calculates the amplitude spectrum of mocap time series.

**syntax**
- `s = mcspectrum(d);`
- `[s f] = mcspectrum(d);`

**input parameters**
- `d`: MoCap structure, norm structure, or segm structure

**output**
- `s`: MoCap structure, norm structure, or segm structure containing amplitude spectra in the `.data` field
- `f`: frequencies in Hz for the frequency channels in the spectra

**examples**

**comments**

**see also**
mcstatmoments

**synopsis**
Calculates the first four statistical moments (mean, standard deviation, skewness, and kurtosis) of data, ignoring missing values.

**syntax**
```matlab
mom = mcstatmoments(d);
```

**input parameters**
- `d`: MoCap data structure, norm data structure, or data matrix.

**output**
- `mom`: structure containing the fields `.mean`, `.std`, `.skewness`, and `.kurtosis`

**examples**

**comments**
Calls the functions `mcmean`, `mcstd`, `mcskewness`, and `mckurtosis`

**see also**
- `mcmean`, `mcstd`, `mcskewness`, `mckurtosis`
mcstd

**synopsis**
Calculates the temporal standard deviation of data, ignoring missing values.

**syntax**
\[ m = \text{mcstd}(d); \]

**input parameters**
\[ d: \text{MoCap data structure, norm data structure, or data matrix.} \]

**output**
\[ m: \text{row vector containing the standard deviations of each data column} \]

**examples**

**comments**

**see also**
\[ \text{mcmean, mcvar, mcskewness, mckurtosis} \]
mctimeder

**synopsis**
Estimates time derivatives of motion capture data. Two options are available, the fast version uses differences between two successive frames and a Butterworth smoothing filter, whereas the accurate version uses derivation with a Savitzky-Golay FIR smoothing filter.

**syntax**

```matlab
d2 = mctimeder(d);
d2 = mctimeder(d, order);
d2 = mctimeder(d, filterparams);
d2 = mctimeder(d, method);
d2 = mctimeder(d, order, filterparams);
d2 = mctimeder(d, order, method);
d2 = mctimeder(d, order, window, method);
```

**input parameters**
- **d**: MoCap structure, norm structure, or segm structure
- **order**: order of time derivative (optional, default = 1).
- **filterparams**: order and cutoff frequency for Butterworth smoothing filter (optional, default [2, 0.2])
- **method**: fast or accurate version; fast version is default, use 'acc' for accurate version (if no window length is given, the default lengths are used, see comment)
- **window**: window length for Savitzky-Golay FIR smoothing filter (optional, default = 7 for first-order derivative)

**output**
- **d2**: MoCap data structure or segm data structure

**examples**

```matlab
d2 = mctimeder(d); % first-order time derivative using the fast method (Butterworth filter with default parameters)
d2 = mctimeder(d, [2 .1]); % first-order time derivative using fast version (second order Butterworth filter with 0.1 Hz cutoff frequency)
d2 = mctimeder(d, 'acc'); % first-order time derivative using the accurate version (Savitzky-Golay filter)
d2 = mctimeder(d, 2, 9, 'acc'); % second-order time derivative with 9-frame window using the accurate version (Savitzky-Golay filter)
```
comments
The default parameters for the Butterworth smoothing filter create a second-order zero-phase digital Butterworth filter with a cutoff frequency of 0.2 Hz.

The window length is dependent on the order of the time derivative and the given window length. It is calculated by \(4n+w-4\). Thus, if the default window length of 7 is used, the window length for the second-order derivative will be 11, and the window length for the third-order derivative will be 15.

For information about the Savitzky-Golay filter, see help sgolayfilt.

The function updates the d.timederorder field as follows: d2.timederorder = d.timederorder + order.

see also
mcsmoothen, mctimeintegr
**mctimeintegr**

**synopsis**
Estimates time integrals of motion capture data using the rectangle rule.

**syntax**
- `d2 = mctimeder(d);`
- `d2 = mctimeintegr(d, order);`

**input parameters**
- `d`: MoCap data structure or segm data structure
- `order`: order of time integral (optional, default = 1)

**output**
- `d2`: MoCap data structure or segm data structure

**examples**
- `d2 = mctimeintegr(d, 2); % second-order time integral`

**comments**
The function updates the `d.timederorder` field as follows: `d2.timederorder = d.timederorder - order`.

**see also**
- `mctimeder`
mctranslate

Synopsis
 Translates motion-capture data by a vector.

Syntax
 d2 = mctranslate(d, transvect);

Input Parameters
 d: MoCap data structure or data matrix
 transvect: translation vector

Output
 d2: MoCap data structure or data matrix

Examples
 d2 = mctranslate(d, [0 1000 0]);

Comments

See also

MoCap Toolbox Manual

mctrim

synopsis
Extracts a temporal section from a MoCap, norm, or segm data structure.

syntax
\[
d2 = \text{mctrim}(d, t1, t2);
\]
\[
d2 = \text{mctrim}(d, t1, t2, \text{timetype});
\]

input parameters
- \(d\): MoCap data, norm, or segm data structure
- \(t1\): start of extracted section
- \(t2\): end of extracted section
- \(\text{timetype}\): either 'sec' (default) or 'frame'

output
- \(d2\): MoCap, norm, or segm data structure containing frames from \(t1\) to \(t2\) (if \(\text{timetype} == \text{\textquote{frame}}\)) or frames between \(t1\) and \(t2\) seconds (if \(\text{timetype} == \text{\textquote{sec}}\)) of MoCap data structure \(d\).

examples
\[
d2 = \text{mctrim}(d, 305, 1506, \text{\textquote{frame}});
\]
\[
d2 = \text{mctrim}(d, 3, 5, \text{\textquote{sec}});
\]

comments

see also
mcvar

**synopsis**
Calculates the variance of data, ignoring missing values.

**syntax**

```matlab
m = mcvar(d);
```

**input parameters**

- **d**: MoCap data structure, norm data structure, or data matrix.

**output**

- **m**: row vector containing the variance of each data column

**examples**

**comments**

**see also**

```
mcmean, mcstd, mcskewness, mckurtosis
```
mcvect2grid

synopsis
Converts a MoCap structure vector to a MoCap structure with three orthogonal views for each component.

syntax
\[ [g, gpar] = mcvect2grid(c, par, dx, dy); \]

input parameters
c: MoCap structure vector
par: animpar structure
dx: horizontal offset between components (default: 2000)
dy: vertical offset between orthogonal views (default: 2000)

output
g: MoCap structure
gpar: animpar structure

examples
\[ [g, gpar] = mcvect2grid(c, par, 1000, 2000); \]

comments

see also
mcwindow

**synopsis**
Performs a windowed time series analysis with a given function.

**syntax**

```matlab
varargout = mcwindow(functionhandle, d);
varargout = mcwindow(functionhandle, d, wlen, hop);
varargout = mcwindow(functionhandle, d, wlen, hop, timetype);
```

**input parameters**
- `functionhandle`: handle to function with which the windowed analysis is performed
- `d`: MoCap data structure or norm data structure
- `wlen`: length of window (optional, default = 2 sec)
- `hop`: hop factor (optional, default = 0.5)
- `timetype`: time type {'sec', 'frame'} (optional, default = 'sec')

**output**
- When used with the functions `mcmean`, `mcstd`, `mcvar`, `mcskewness`, and `mckurtosis`, the output is a two-dimensional matrix where the first index corresponds to window number and the second index to marker/dimension.
- When used with `mcperiod`, the function returns four output parameters `[per, ac, eac, lag]`, where `per` is a two-dimensional matrix with the first index corresponding to window number and the second to marker/dimension. Output parameters `ac` and `eac` are three-dimensional matrices, with the first index corresponding to window number, the second to lag, and the third to marker/dimension. The output parameter `lag` is a vector containing the lag values for the autocorrelations.

**examples**

```matlab
stds = mcwindow(@mcstd, d, 3, 0.5);
[per, ac, eac, lags] = mcwindow(@mcperiod, d);
```

**comments**

**see also**
- `mcmean`, `mcstd`, `mcvar`, `mcskewness`, `mckurtosis`, `mcperiod`
Never confuse motion with action.

— Benjamin Franklin —