Table of Contents
MoCap Toolbox Manual

Foreword ................................................................. 5
Acknowledgments ...................................................... 5

Release Notes .......................................................... 7
Version 1.5..................................................................... 7
New functions............................................................ 7

Introduction .................................................................. 9

General ...................................................................... 12
Functions .................................................................... 13
Data Structures.......................................................... 17
Parameter structures..................................................... 18
Add-ons and extensions ................................................ 19
  Periodic quantity of motion ......................................... 19
  Realtime streaming of mocap data .................................. 19

Examples .................................................................... 21
Reading, Editing, and Visualizing MoCap Data (mcdemo1).................. 22
Transforming MoCap data (mcdemo2)........................................ 32
Kinematic analysis (mcdemo3).............................................. 36
Time-series analysis (mcdemo4)............................................. 42
Kinetic analysis (mcdemo5)................................................ 45
Creating animations (mcdemo6 - mcdemo9)............................. 51
  Basics (mcdemo6)......................................................... 51
  Merging data for animations (mcdemo7)............................. 52
  Colored animations (mcdemo8)........................................ 53
  Perspective Projection (mcdemo9)...................................... 55
Principal Components Analysis (mcdemo10)............................... 58
Analyzing Wii data (mcdemo11)............................................ 59

Data and Parameter Structure Reference .......................... 63
MoCap data structure..................................................... 64
norm data structure ....................................................... 65
segm data structure ....................................................... 66
m2jpar parameter structure.............................................. 67
j2spar parameter structure .............................................. 68
animpar parameter structure ............................................ 69

Function Reference .................................................. 71
mc2frontal........................................................................ 72
mcaddframes................................................................. 73
mcanimate ..................................................................... 74
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 Dominique de Beul for the providing the bvh parser.

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

Thanks to Kristian Nymoen for providing the function mcmocapgram and the real-time streaming.

Thanks to Roberto Rovegno for contributing to the mewritetsv function.

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

Thanks to Federico Visi for providing the mcpqom, meplotqom, and merepovizz functions and contributing to the mesort function.
Thanks to Dominique de Beul, Michiel Demey, Frank Desmet, Tommi Himberg, Herbert Jäger, Alexander Refsum Jensenius, Luiz Naveda, Kristian Nymoen, and Erwin Schoonderwaldt for reporting bugs in the toolbox and suggesting improvements.
For new features and bug fixes done in previous versions, please refer to the releasenotes_v1.5.txt included in the toolbox release.

Version 1.5

New functions
mcaddframes: duplicates frames
mccomplexity: calculates the complexity of movement based on entropy of the first principal component
mcfluidity: calculates the fluidity/circularity of mocap data
mcrepovizz: exports MoCap structure as repoVizz files
mcreverse: reverses dimensions of motion-capture data
mcrotationrange: calculates the rotation range between two markers
mcsetlength: sets mocap data to the length given
mcsort: sorts mocap data according to marker names

Bug fixes
meanimate, mcplotframe, mcinitanimpar: direct video file making; changing use of projection parameters; returning from function with setting video parameters, but without creating video
mcodemodata: updated animpar variables to fit new animation parameter structure
mcdemo6-mcdemo9: fit new animation parameter structure
mcgetmarkername: now for norm data as well
mchilbert: keep data structure same as input file, added flag for indicating phase wrap
mcinitstruct: fixed inconsistency in naming in the manual
mcmerge: fixing animation parameter structure merging
memocapgram: norm data included

mcplotframe: move axes definition outside the main plotting loop for efficiency

mCREAD: added bvh support, added potential troubleshoot for c3d data

mcreadc3d: check for matching frame no and data size (Optitrack issue)

mes2j: runtime efficiency

mctimeintegr: for norm data as well

readc3d: changed machinetype parameter - change back as indicated in script if you run into issues reading in c3d files
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. The latest implementations and developments have been made on Matlab version 8.4 (R2014b) running on Macintosh OS X v10.10.

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)
- For reading in .bvh files: http://staffwww.dcs.shef.ac.uk/people/N.Lawrence/mocap/ (mocap and ndlutil toolboxes – also the github versions work)

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 mocap-toolbox@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 64 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>Function</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcread</td>
<td>read MoCap data files</td>
</tr>
<tr>
<td>mcreademg</td>
<td>read emg files in tsv format</td>
</tr>
<tr>
<td>mcmissing</td>
<td>report missing frames and markers</td>
</tr>
<tr>
<td>mctrim</td>
<td>extract a temporal segment from MoCap data</td>
</tr>
<tr>
<td>mccut</td>
<td>cut two MoCap structures to the length of the shorter one</td>
</tr>
<tr>
<td>mcaddframes</td>
<td>duplicate frames</td>
</tr>
<tr>
<td>mcsetlength</td>
<td>set MoCap data to the length given</td>
</tr>
<tr>
<td>mcsmoothen</td>
<td>smoothen MoCap data</td>
</tr>
<tr>
<td>mcmerge</td>
<td>merge two MoCap data structures</td>
</tr>
<tr>
<td>mcsort</td>
<td>sorts MoCap data according to marker names</td>
</tr>
<tr>
<td>mcgetmarker</td>
<td>extract a subset of markers from MoCap data</td>
</tr>
<tr>
<td>mcsetmarker</td>
<td>replace a subset of markers</td>
</tr>
<tr>
<td>mcconcatenate</td>
<td>concatenate markers from different MoCap or norm data structure</td>
</tr>
<tr>
<td>mcgetmarkername</td>
<td>get names of markers from MoCap data</td>
</tr>
<tr>
<td>mcfillgaps</td>
<td>fill missing data</td>
</tr>
<tr>
<td>mcinitstruct</td>
<td>initialize MoCap or norm data structure</td>
</tr>
<tr>
<td>mcreorderdims</td>
<td>reorder the Euclidean dimensions in the MoCap data</td>
</tr>
<tr>
<td>mcreverse</td>
<td>reverse dimensions of MoCap data</td>
</tr>
<tr>
<td>mcresample</td>
<td>resample MoCap data</td>
</tr>
<tr>
<td>mcrepovizz</td>
<td>export MoCap structure as repoVizz files</td>
</tr>
<tr>
<td>mcc3d2tsv</td>
<td>convert a c3d file into a tsv file</td>
</tr>
<tr>
<td>mcwritetsv</td>
<td>save MoCap structure as tsv file</td>
</tr>
<tr>
<td>mccenter</td>
<td>center MoCap data to have a mean of [0 0 0]</td>
</tr>
<tr>
<td>mctranslate</td>
<td>translate MoCap data</td>
</tr>
<tr>
<td>mcrotate</td>
<td>rotate MoCap data</td>
</tr>
<tr>
<td>Coordinate System Conversion</td>
<td>mc2frontal</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>mcvect2grid</td>
</tr>
<tr>
<td>Kinematic Analysis</td>
<td>mcinitm2jpar</td>
</tr>
<tr>
<td></td>
<td>mcm2j</td>
</tr>
<tr>
<td></td>
<td>mcinitj2spar</td>
</tr>
<tr>
<td></td>
<td>mcj2s</td>
</tr>
<tr>
<td></td>
<td>mcs2j</td>
</tr>
<tr>
<td></td>
<td>mcs2posture</td>
</tr>
<tr>
<td></td>
<td>mcnorm</td>
</tr>
<tr>
<td></td>
<td>mctimedder</td>
</tr>
<tr>
<td></td>
<td>mctimeintegr</td>
</tr>
<tr>
<td></td>
<td>mccumdist</td>
</tr>
<tr>
<td></td>
<td>mcmarkerdist</td>
</tr>
<tr>
<td></td>
<td>mcboundrect</td>
</tr>
<tr>
<td></td>
<td>mccomplexity</td>
</tr>
<tr>
<td></td>
<td>mcfluidity</td>
</tr>
<tr>
<td></td>
<td>mcrotationrange</td>
</tr>
<tr>
<td></td>
<td>mcsegmangle</td>
</tr>
<tr>
<td></td>
<td>mcperiod</td>
</tr>
<tr>
<td></td>
<td>mcdecompose</td>
</tr>
<tr>
<td></td>
<td>mcspectrum</td>
</tr>
<tr>
<td>Kinetic Analysis</td>
<td>mcgetsegmpar</td>
</tr>
<tr>
<td></td>
<td>mckinenergy</td>
</tr>
<tr>
<td>TIME-SERIES ANALYSIS</td>
<td><code>mcpotenergy</code></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td><code>mcmean</code></td>
</tr>
<tr>
<td></td>
<td><code>mcstd</code></td>
</tr>
<tr>
<td></td>
<td><code>mcvar</code></td>
</tr>
<tr>
<td></td>
<td><code>mcskewness</code></td>
</tr>
<tr>
<td></td>
<td><code>mckurtosis</code></td>
</tr>
<tr>
<td></td>
<td><code>mcstatmoments</code></td>
</tr>
<tr>
<td></td>
<td><code>mcwindow</code></td>
</tr>
<tr>
<td>VISUALIZATION</td>
<td><code>mcplottimeseries</code></td>
</tr>
<tr>
<td></td>
<td><code>mcplotphaseplane</code></td>
</tr>
<tr>
<td></td>
<td><code>mcinitanimpar</code></td>
</tr>
<tr>
<td></td>
<td><code>mccreateconnmatrix</code></td>
</tr>
<tr>
<td></td>
<td><code>mcplotframe</code></td>
</tr>
<tr>
<td></td>
<td><code>mcanimate</code></td>
</tr>
<tr>
<td></td>
<td><code>mcsimmat</code></td>
</tr>
<tr>
<td></td>
<td><code>mcmocapgram</code></td>
</tr>
<tr>
<td>PROJECTION</td>
<td><code>mcpcaproj</code></td>
</tr>
<tr>
<td></td>
<td><code>mcicaproj</code></td>
</tr>
<tr>
<td></td>
<td><code>mcsethares</code></td>
</tr>
<tr>
<td></td>
<td><code>mceigenmovement</code></td>
</tr>
<tr>
<td>OTHER</td>
<td><code>mcbandpass</code></td>
</tr>
<tr>
<td></td>
<td><code>mchilbert</code></td>
</tr>
<tr>
<td></td>
<td><code>mchilberthuang</code></td>
</tr>
<tr>
<td></td>
<td><code>mcfilteremg</code></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 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 .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 mctimedér and 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 mcm2j 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 mcnorm, is similar to the MoCap data structure, except that its .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 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.

**Add-ons and extensions**

**Periodic quantity of motion**

Functions that estimate and plot the periodic quantity of motion have been implemented by Federico Visi and Rodrigo Schramm. These functions can be downloaded from the mocap toolbox download page, section “Extensions”. To use the function, download the zip-folder and extract it either to the toolbox folder or to another folder that your Matlab distribution can access. A demo explaining the use of the functions is included in the package. For more information and support please contact Federico or Rodrigo and have a look at the following publication:


**Realtime streaming of mocap data**

A solution for realtime streaming of mocap data from the mocap toolbox has been implemented by Kristian Nymoen. It allows playback of synchronized sound and motion capture data, as well as looping, scrubbing, etc. You can visualise the data in 3D while interacting with the visualisation.

The visualisation and GUI is implemented in Max 6, and a standalone application for Mac is also included (meaning that it is not necessary to have Max installed). The files can be downloaded from http://www.fourms.uio.no/software/mcrtanimate/.

If somebody wants to have this running on Windows, it should work in theory, but you will have to have Max installed, and download the necessary dependencies (listed in the readme file).
The implementation requires the Instrument Control Toolbox for Matlab (http://www.mathworks.se/products/instrument/). A workaround without this Toolbox is under development and will be provided soon.

This is still a prototype implementation, so please provide feedback (suggestions, new features, ...) to Kristian.
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* format produced by the Qualisys motion capture system, the *.mat* format produced by the Qualisys motion capture system, and the *.wii* file format, produced by the WiiDataCapture software (available at [www.jyu.fi/music/coe/materials/mocaptoolbox](http://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.

```
load mcdemodata
whos
```

```
Name      Size      Bytes  Class      Attributes
---       ----      ------  ----       --------
dance1    1x1       1013572 struct
dance2    1x1       1013572 struct
j2spar    1x1       2168    struct
japar     1x1       2530    struct
m2jpar    1x1       3464    struct
mapar     1x1       2914    struct
walk1     1x1       241862  struct
walk2     1x1       275474  struct
wiidata   1x1       45560   struct
```

Variable `walk1` is a *MoCap data structure*:

```
walk1
```
walk1 =
    type: 'MoCap data'
    filename: '28-Karolien-Walking.tsv'
    nFrames: 351
    nCameras: 8
    nMarkers: 28
    freq: 60
    nAnalog: 0
    anaFreq: 0
timerOrder: 0
    markerName: {28x1 cell}
        data: [351x84 double]
    analogdata: []
    other: [1x1 struct]

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

    [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')

![Graph showing missing data](image)

Markers 2 and 6 have missing frames. The missing data can be filled using the function mcfillgaps:
walk1 = mcfillgaps(walk1, 100);

[mf, mm, mgrid] = mcmissing(walk1);
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')

The variable walk1 has no more missing frames.

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

mcplottimeseries(walk1, 3, 'dim', 3) % marker 3, dimension 3
mcplottimeseries(walk1, [4 8 12], 'dim', 3, 'timetype', 'frame')
% markers 4, 8, & 12, dimension 3, frames on x-axis

mcplottimeseries(walk1, [4 8 12], 'dim', 3, 'plotopt', 'comb')
% markers 4, 8, & 12, dimension 3, combined into one plot

mcplottimeseries(walk1, [4 8 12], 'dim', 3, 'label', 'mm', 'names', 1)
% markers 4, 8, & 12, dimensions 1:3, label on y-axis set, marker names instead of numbers
mcplottimeseries(walk1, {'Head_BR', 'Sternum', 'Hip_BR'}, 'dim', 1:3)
% using marker names instead of numbers in function call, dimensions 1:3

Marker locations in single frames can be plotted using the function `mcplotframe`. This function plots the (x,z) projection of the markers:

```matlab
mcplotframe(walk1, 160);
```
Because the parameter structure was not given in the previous call, the function used the default settings for the animation parameters:

```matlab
mcinitanimpar
ans =
```
```matlab
type: 'animpar'
screds: [800 600]
limits: []
  az: 0
  el: 0
msize: 12
colors: 'kwwww'
markercolors: []
conncolors: []
tracecolors: []
numbercolors: []
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'
scresize: [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
output: ‘tmp’
videoformat: ‘avi’
createframes: 0
getparams: 0
perspective: 0
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 27 28
 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 screen size:

mapar.scrsize = [800 600];

Using the parameter structure mapar, we get the following visualization:
We can add marker numbers to the plot by setting the field `.showmnum` to have the value 1:

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

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

```matlab
mapar.colors='wbgrc';
mcpplotframe(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:

```plaintext```
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:

```plaintext```
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:

```
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:
d1rot1 = mcrotate(dance1, 90, [0 0 1]);
mcplotframe(d1rot1, 50, mapar);

Next, let us rotate the data in dance1 by 90 degrees counterclockwise around the x axis:

d1rot2 = mcrotate(dance1, 90, [1 0 0]);
mcplotframe(d1rot2, 50, mapar);

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

d1rot3 = mcrotate(dance1, 90, [0 1 0]);
mcplotframe(d1rot3, 50, mapar);
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 \textit{dance1} and \textit{dance2} and merge them for visualization

\begin{verbatim}
d1 = mctrim(dance1, 0, 2);
d2 = mctrim(dance2, 0, 2);
d2 = mctranslate(d2, [2000 0 0]);
[d, par] = mcmerge(d1, d2, mapar, mapar);
mcpplotframe(d, 60, par);
\end{verbatim}

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

\begin{verbatim}
mcpplotframe(d, 1:10:71, par);
\end{verbatim}
(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`:

```
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):

```
mcplottimeseries(d2v, [1 19 25], 'dim', 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:

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

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

```matlab
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:
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
```

```
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:

```
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:

```
[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

```
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
imagesc(eac(:,:,3)), axis xy
set(gca,'XTick',0:4:46)
set(gca,'XTickLabel',0.5*(0:4:46))
set(gca,'YTick',[0 30 60 90 120])
set(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:

```
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.

```matlab
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)
```

43
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 \texttt{mcm2j}. The parameters needed for this conversion are in the variable \texttt{m2jpar}:

\begin{verbatim}
m2jpar
m2jpar =

    type: 'm2jpar'
    nMarkers: 20
    markerNum: {1x20 cell}
    markerName: {1x20 cell}
\end{verbatim}

The information concerning which markers correspond to each joint is contained in the field \texttt{m2jpar.markerNum}. The names of the new joints are in \texttt{m2jpar.markerName}. For instance, ...

\begin{verbatim}
m2jpar.markerName{1}
an =
root

m2jpar.markerNum{1}
an =
    9  10  11  12
\end{verbatim}

... 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:

\begin{verbatim}
walk2j = mcm2j(walk2, m2jpar)
\end{verbatim}
walk2j =
    type: 'MoCap data'
    filename: '25-Walking.tsv'
    nFrames: 401
    nCameras: 8
    nMarkers: 20
    freq: 60
    nAnalog: 0
    anaFreq: 0
    timerOrder: 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`:

    japar = 
    type: 'animpar'
    scrsize: [400 300]
    limits: []
        az: 0
        el: 0
    msize: 6
        colors: 'kwwww'
    markercolors: []
    conncolors: []
    tracecolors: []
    numbercolors: []
        cwidth: 1
        twidth: 1
        conn: [19x2 double]
    conn2: []
    trm: []
    trl: 0
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`:

```matlab
j2spar =
    type: 'j2spar'
  rootMarker: 1
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`:

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

The second argument in the function call, `segmindex`, associates each joint in `walk1j` and `walk1s` with a segment type. The numbers refer to the distal joint of the respective segment. Joints that are not distal to any segment have zero values. 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'.

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:

```
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:

\[ \text{[trans, rot]} = \text{mckinenergy(walk2j, walk2s, spar)}; \]

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 either produce an animation video (.avi or .mp4 format) or animation frames (single .png files) with the MoCap toolbox. The frames would 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'
    scrsiz: [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
```
Let us change the frames-per-second value to 15

```
mapar.fps = 15;
```

The animation will be stored into the current directory with the filename `tmp.avi`. If needed, the current directory should be changed before creating the animation.

The animation is produced as follows:

```
newpar = mcanimate(walk2, mapar);
```

If you wish to plot consecutive frames (png files) instead of creating a video file, set the animation parameter `createframes` to 1:

```
mapar.createframes = 1;
```

You will then find the png files in a folder called `tmp` in the current directory.

**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 for the file:

```matlab
mapar.fps = 15;
mapar.output = '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`:

```matlab
[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.

```matlab
newpar = mcanimate(d, par);
```

There should be now a file called `twodancers.avi` in the current directory.

**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: 'kwww'
markercolors: []
conncolors: []
tracecolors: []
numbercolors: []
cwidth: 1
twidth: 1
conn: [43x2 double]
conn2: []
trm: []
```
Let us change the frames-per-second value to 15

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

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)

\[ \text{mapar.markercolors} = 'bwwgwrwwwwwywwwwwwmmwcbwg'; \]

and let us have a look at the new colors:

\[ \text{mcplotframe(dance2, 150, mapar);} \]

Now let us set the markers that we want to trace and the trace length (in seconds):

\[ \text{mapar.trm} = [1 6 12 19 21 24]; \]
mapar.trl=3;
And let us set individual colors for the traces:

mapar.tracecolors='grymcb';
We rotate the figure to be frontal on average

dance2=mc2frontal(dance2,9,10);
Now we make the animation:

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 dance2:

Perspective Projection (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 mcdemodata and change a couple of parameters of the animpar structure mapar

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 `mapar`, so that the walker will walk towards us

```matlab
mapar.az=270;
```

We do not want to create a video this time, but have a look at the separate frames (to better see the differences in the projection), we set the createframes parameter accordingly and re-name the default file name, which will serve as the folder name now, into which the frames (i.e., png files) are saved

```matlab
mapar.createframes = 1;
mapar.output = 'pers0';
```

Now we create an animation out of that:

```matlab
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:

![Frame 10](image1.png)
![Frame 20](image2.png)
![Frame 40](image3.png)

The idea of the perspective projection is to visualize that the figure is actually walking forward. To activate the perspective projection, we set the perspective parameter in the animpar structure to 1 and call the `mcanimate` function again:

```matlab
mapar.perspective = 1;
mapar.output = 'pers1';
mcanimate(walk2, mapar);
```

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`.

```matlab
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.

```matlab
mapar.pers.c = [1000  -4000  1000];
mapar.output = 'pers2';
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,

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

and convert it into a joint representation

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

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

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

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

```
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:

```
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

```
s=mcj2s(j,j2spar); % convert to segment structure
```
[pcs,ps]=mcpcaproj(s,1:3); % perform PCA
for k=1:3 % convert PC projections back to joint structures
cpj(k) = mcs2j(pcs(k), j2spar);
end

Next, let us plot the first three PC projections:

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.

In the file mcdemodata, the variable \( \text{wiidata} \) contains acceleration data captured using the Nintendo Wii controller and the WiiDataCapture software:

```matlab
load mcdemodata
```
As the field `.timederOrder` indicates, this variable holds acceleration data. Let us plot the third (vertical) component of the acceleration data:

```matlab
mcplottimeseries(wiidata,1, 'dim', 3)
```

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

```matlab
wd2 = mcsmoother(wiidata,25);
mcplottimeseries(wd2,1,'dim',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:

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

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:

```
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
- **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)
- **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
- **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 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
- **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)
- **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 -1 0) 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. That means the first segment is empty.

**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’)
scrsise: 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 xzmin xzmax]
az: azimuth angle (in degrees) of viewing point
el: elevation angle (in degrees) of viewing point
msize: size of markers
colors: five-character string or RGB triple (5x3) containing the colors of background, markers, connections, traces, and marker numbers, respectively
markercolors: string or RGB triple (nx3) containing the individual colors for the markers
conncolors: string or RGB triple containing the individual colors for the connector lines
tracecolors: string or RGB triple containing the individual colors for the trace lines
numbercolors: string or RGB triple 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 animate function before it calls the mcplotframe function
fps: frames per second used in animation
output: either file name for video file, of folder for pgn frames (‘tmp’)
videoformat: specifies video file format, either ‘avi’ or ‘mpeg4’ (‘avi’)
createframes: create png frames instead of video file, 1=frames, 0=video file (0)
getparams: return animation parameters, without plotting or animating frames, 1=yes, 0=no (0)
perspective: perform perspective projection, 0 = orthographic (default), 1 = perspective (0)
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
mcinitanimpar
Function Reference
mc2frontal

**synopsis**
Rotates MoCap 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
mcaddframes

**synopsis**
Duplicates frames in a given mocap structure; either last frame in the end, first frames in the beginning, or at a given position in the middle.

**syntax**
```
d2 = mcaddframes(d, add);
d2 = mcaddframes(d, add, 'timetype', 'frame', 'location', 'beginning');
d2 = mcaddframes(d, add, 'location', 'middle', 'position', n);
```

**input parameters**
- `d`: MoCap or norm data structure
- `add`: total amount of frames to be added
- `timetype`: amount of frames given in frames ('frame') or seconds ('sec') (default: sec)
- `location`: location where frames are added: 'beginning', 'middle', or 'end' (default: end)
- `position`: position where frames are added – only needed when using 'middle' as location. The timetype parameter applies to both add and position.

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

**examples**
```
d2 = mcaddframes(d, 60);
d2 = mcaddframes(d, 60, 'location', 'middle', 100);
```

**comments**
- timetype, location, and position are optional. Default values are used if not specified.

**see also**
mcanimate

synopsis
Creates animation of mocap data and saves it to file (.avi or mpeg-4) or as consecutive frames (.png). Matlab's VideoWriter function is used to create the video file.

COMPATIBILITY NOTES (v. 1.5): The 'folder'-field (animpar structure, v. 1.4) has been changed to 'output' and is used as file name for the animation (and stored to the current directory) or as folder name in case frames are to be plotted.
Please use the function without the projection input argument, but specify it in the animation structure instead.

syntax
par = mcanimate(d);
par = mcanimate(d, par);

input parameters
d: MoCap data structure
par: animpar structure (optional)

output
par: animpar structure used for plotting the frames

examples
mcanimate(d, par);

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**
```matlab
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**
```matlab
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**

\[
\begin{align*}
\text{br} & = \text{mcboundrect}(d); \\
\text{br} & = \text{mcboundrect}(d, \text{mnum}); \\
\text{br} & = \text{mcboundrect}(d, \text{mnum, w, hop});
\end{align*}
\]

**input parameters**
- `d`: MoCap data structure
- `mnum`: marker numbers (optional; if no value given, all markers are used)
- `w`: length of analysis window (optional; default: 4 sec)
- `hop`: overlap of analysis windows (optional; default: 2 sec)

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

**examples**

\[
\begin{align*}
\text{br} & = \text{mcboundrect}(d); \\
\text{br} & = \text{mcboundrect}(d, [1~3~5]); \\
\text{br} & = \text{mcboundrect}(d, [1:d.nMarkers], 3, 1);
\end{align*}
\]

**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**

**references**
mcc3d2tsv

synopsis
Converts a c3d file into a tsv file.

syntax
mcc3d2tsv(fn, path);

input parameters
fn: name of c3d file
path: path to save the tsv file (optional). If no path is given, file is saved to current directory

output
tsv file, saved in the current or in the specified directory

examples
% mcc3d2tsv('file.c3d')
% mcc3d2tsv('file.c3d', 'folder')
% mcc3d2tsv('file.c3d', '/path/folder') %(Mac)

comments

see also
mcread
mccenter

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

**syntax**
```matlab
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**
mccomplexity

**synopsis**
Calculates the complexity of movement based on entropy of the proportion of variance contained in the principal components. A high value indicates a high complexity, whereas a low value indicated low complexity.

**syntax**
c = mccomplexity(d, mnum);

**input parameters**
d: MoCap data structure
mnum: marker numbers (optional; if no value given, all markers are used)

**output**
c: complexity value, between 0 and 1

**examples**
c = mccomplexity(d);
c = mccomplexity(d, 4:7);

**comments**
Data will be filled in case of missing frames.

**see also**
mecapproj

**references**
mcconcatenate

**synopsis**
Concatenates markers from different MoCap or norm data structure.

**syntax**
```matlab
d2 = mcconcatenate(d1, mnum1, d2, mnum2, d3, mnum3, ...);
```

**input parameters**
- `d1, d2, d3, ...`: MoCap or norm data structure
- `mnum1, mnum2, mnum3, ...`: vector containing the numbers of markers to be extracted from the preceding MoCap structure

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

**examples**
```matlab
d2 = mcconcatenate(d1, [1 3 5], d2, [2 4 6]);
d2 = mcconcatenate(d1, 1, d2, 2, d1, 3, d3, 4, d2, 5);
```

**comments**
Each mocap structure must have a corresponding marker number or number array.
All mocap structures must have identical frame rates.
If the numbers of frames are not equal, the output MoCap structure will be as long as the shortest input MoCap structure.

**see also**
- `mcgetmarker`, `mcmerge`
**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**
\[d11, d22] = mccut(d1, d2);

**input parameters**
d1, d2: MoCap or norm structures

**output**
d11, d22: MoCap or norm structures, one shortened and one original (both with same number of frames)

**examples**

**comments**

**see also**
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 (timerOrder = 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**
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
\[ d2 = \text{mcfillgaps}(d); \]
\[ d2 = \text{mcfillgaps}(d, \text{maxfill}); \]
\[ d2 = \text{mcfillgaps}(d, \text{method}); \]
\[ d2 = \text{mcfillgaps}(d, \text{maxfill}, \text{method}); \]

input parameters
\(d\): MoCap, norm, or segm data structure
\(\text{maxfill}\): maximal length of gap to be filled in frames (optional, default = 1000000)
\(\text{method}\): 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
\[ d2 = \text{mcfillgaps}(d); \]
\[ d2 = \text{mcfillgaps}(d, 120); \]
\[ d2 = \text{mcfillgaps}(d, 'nobefill'); \]
\[ d2 = \text{mcfillgaps}(d, 60, 'fillall'); \]

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

see also
mcfilteremg

**synopsis**
Filters EMG data.

**syntax**
```matlab
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**
```matlab
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
mcfluidity

**synopsis**
Calculates the fluidity/circularity of mocap data, defined as the ratio between velocity and acceleration of the normed and averaged mocap data.

**syntax**
f = mcfluidity(d, mnum);

**input parameters**
d: MoCap data structure
mnum: marker numbers (optional; if no value given, all markers are used)

**output**
f: fluidity value (the higher the value, the higher the smoothness/fluidity)

**examples**
f = mcfluidity(d, 4:6);

**comments**

**see also**

**references**
mcgetmarker

**synopsis**
Extracts a subset of markers.

**syntax**
\[
d2 = mcgetmarker(d, mnum);
\]

**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**
\[
d2 = mcgetmarker(d, [1 3 5]);
\]

**comments**

**see also**
- mcsetmarker, mcconcatenate
mcgetmarkernames

**synopsis**
Returns the names of markers.

**syntax**
```
mn = mcgetmarkernames(d);
```

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

**output**
mn: cell structure containing marker names

**examples**

**comments**

**see also**
mcgetsegmpar

synopsis
Get parameters for body segments.

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

**input parameters**
- `d`: MoCap or norm data structure
- `wrap`: flag to indicate if phase is returned as wrapped or unwrap (default: unwrapped); 0 or empty: unwrap, 1: wrap

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

**examples**
```matlab
amp = mchilbert(d);
[amp, phase, h] = mchilbert(d, 1);
```

**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**
```
ap = mcinitanimpar;
```

**input parameters**
(none)

**output**
ap: animation parameter (animpar) structure

**examples**

**comments**
See also description of the animpar structure (default values given in parentheses:)
- scrsize: 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') or RGB triplet (5x3)
- markercolors: String holding marker colors ([]) or RGB triplet
- conncolors: String holding connector (line) colors ([]) or RGB triplet
- tracecolors: String holding trace colors (only animations) ([]) or RGB triplet
- numbercolors: String holding number colors (indicated in the numbers array) ([]) or RGB triplet
- 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: create animation, 1=yes, 0=no (0)
- fps: frames per second for animation (30)
- output: either file name for video file, of folder for pgn frames (‘tmp’)
videoformat: specifies video file format, either 'avi' or 'mpeg4' ('avi')
createframes: create png frames instead of video file, 1=frames, 0=video file (0)
getparams: return animation parameters, without plotting or animating frames, 1=yes, 0=no (0)
perspective: perform perspective projection, 0 = orthographic (default), 1 = perspective (0)
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]

Colors can be given as strings if only the MATLAB string color options are used. However, any color can be specified by using RGB triplets - for example, plotting the first two markers in gray: par.markercolors=[.5 .5 .5; .5 .5 .5];

see also
mccreateconnmatrix, mcplotframe, mcanimate
mcinitj2spar

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

**syntax**
```matlab
par = mcinitj2spar;
```

**input parameters**
(None)

**output**
- `par`: 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**

```matlab
d1 = mcinitstruct;
d1 = mcinitstruct(type);
d1 = mcinitstruct(type, data);
d1 = mcinitstruct(type, data, freq);
d1 = mcinitstruct(type, data, freq, markerName);
d1 = mcinitstruct(type, data, freq, markerName, fn);
d1 = mcinitstruct(data, freq);
d1 = mcinitstruct(data, freq, markerName);
d1 = mcinitstruct(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**

```matlab
d1 = mcinitstruct;
d1 = mcinitstruct('norm data', data);
d1 = mcinitstruct(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: []
analoggdata: []
other:
    other.desc: 'DESCRIPTION --'
    other.timeStamp: 'TIME_STAMP--'
    other.dataIncluded: '3D'

see also
mcj2s

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

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

**input parameters**
- \( d \): MoCap data structure
- \( par \): 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
\[\text{[te, re]} = \text{mckinenergy}(d, \text{segd}, \text{spar});\]

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
\begin{align*}
\text{segd} &= \text{mcj2s}(d, j2spar); \\
\text{spar} &= \text{mcgetsegmpar('Dempster', segmnum)}; \\
\text{[te, re]} &= \text{mckinenergy}(d, \text{segd}, \text{spar});
\end{align*}

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**
\[ d2 = \text{mcm2j}(d, \text{par}); \]

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

**output**
- d2: MoCap data structure

**examples**

**comments**
The fields 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**
```
dist = mcmarkerdist(d, m1, m2);
```

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

**output**
- `dist`: column vector

**examples**
```
dist = mcmarkerdist(d, 1, 5);
```

**comments**

**see also**
mcmean

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

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

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 or norm data structures
 p1, p2: animpar structures for d1 and d2

 output
 d3: MoCap or norm 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. All animation parameters will be taken from the first animpar file, apart from any color, marker, trace definition, and connection matrices.

 see also
 mcconcatenate
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**
**mcmocapgram**

**synopsis**
Plots mocapgram (shows positions of a large number of markers as projection onto a color-space).

**syntax**
```
h = mcmocapgram(d);
mcmocapgram(d);
mcmocapgram(d, timetype);
```

**input parameters**
- `d`: MoCap or norm data structure.
- `timetype`: time type used in the plot ('sec' (default) or 'frame')

**output**
- `h`: figure handle

**examples**
```
mcmocapgram(d,'frame');
h = mcmocapgram(d);
```

**comments**

**see also**
mcnorm

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

**syntax**
- `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**
- `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
\[
[dp, p] = \text{mcpcaproj}(d); \\
[dp, p] = \text{mcpcaproj}(d, pc); \\
[dp, p] = \text{mcpcaproj}(d, pc, proj);
\]

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

comments

see also
mcicaproj, mcsethares

references
mcperiod

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

**syntax**

\[
[\text{per}, \text{ac}, \text{eac}, \text{lag}] = \text{mcperiod}(d);
\]

\[
[\text{per}, \text{ac}, \text{eac}, \text{lag}] = \text{mcperiod}(d, \text{maxper});
\]

\[
[\text{per}, \text{ac}, \text{eac}, \text{lag}] = \text{mcperiod}(d, \text{method});
\]

\[
[\text{per}, \text{ac}, \text{eac}, \text{lag}] = \text{mcperiod}(d, \text{maxper}, \text{method});
\]

**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**

\[
[\text{per}, \text{ac}, \text{eac}, \text{lag}] = \text{mcperiod}(d, 3);
\]

\[
\text{per} = \text{mcperiod}(d, \text{'highest'});
\]

**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.

COMPATIBILITY NOTES (v. 1.5): Please use the function without the projection input argument, but specify it in the animation structure instead.

**syntax**
```
par = mcplotframe(d, n);
par = mcplotframe(d, n, par);
```

**input parameters**
- `d`: MoCap data structure
- `n`: vector containing the numbers of the frames to be plotted
- `par`: animpar structure (optional)

**output**
- `par`: animpar structure used for plotting the frames (if color strings were used, they will converted to RGB triplets)

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

**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.

**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, [3 5 7], '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

**synopsis**
Plots motion capture data as time series. NEW SYNTAX IN VERSION 1.3.1

**syntax**
- `mcplottimeseries(d, marker)` % for MoCap or norm data structure
- `mcplottimeseries(d, marker, 'dim', dim)` % specifying dimensions
- `mcplottimeseries(d, marker, 'timetype', timetype)` % axis unit
- `mcplottimeseries(d, marker, 'plotopt', plotopt)` % combined or separate plots
- `mcplottimeseries(d, marker, 'label', label)` % y-axis label
- `mcplottimeseries(d, marker, 'names', names)` % marker names
- `mcplottimeseries(s, segm, 'var', var)` % for segm data structure

**input parameters**
- `d/s`: MoCap data structure, norm data structure, or segm data structure
- `marker`: vector containing marker numbers or cell array containing marker names (for MoCap or norm data structure)
- `segm`: body segment numbers or cell array containing segment names (for segm data structure)
- `dim`: dimensions to be plotted (for MoCap data structure - default: 1)
- `var`: variable to be plotted for segment segm (for segm data structure - default: 1)
- `timetype`: time type used in the plot ('sec' (seconds - default) or 'frame')
- `plotopt`: plotting option (for MoCap or norm data structure); 'sep' (default) or 'comb':
  - `sep`: all time series are plotted in separate subplots
  - `comb`: all time series will be plotted into the same plot using different colors
- `label`: y-axis label (default: no y-axis label). X-axis label is always set, according to timetype (however, for plotting neither x-axis nor y-axis labels: 'label', 0)
- `names`: if marker names (instead of numbers) are plotted in title and legend (0: numbers (default), 1: names)

**output**
Figure.

**examples**
- `mcplottimeseries(d, 2)` % MoCap or norm data structure, marker 2, dim 1
- `mcplottimeseries(d, {'Head_FL','Finger_L'})` %marker names instead of numbers (works for segments as well)
- `mcplottimeseries(d, 1:3, 'dim', 1:3)` % markers 1 to 3, dimensions 1 to 3
- `mcplottimeseries(d, 1:3, 'dim', 3, 'timetype', 'frame')` % frames as x axis unit
mcplottimeseries(d, 5, 'dim', 1:3, 'plotopt', 'comb') % all in one plot, different colors per dim
mcplottimeseries(d, 5, 'dim', 1:3, 'plotopt', 'comb', 'label', 'mm') % y-axis label: mm
mcplottimeseries(d, 5, 'dim', 1:3, 'timetype', 'frame', 'label', 0) % no x-axis (and no y-axis) label
mcplottimeseries(d, 5, 'names', 1) % marker names (instead of numbers) plotted in title and legend
mcplottimeseries(s, [3 6 20], 'var', 'angle') % for segm data structure
mcplottimeseries(s, 5:10, 'var', 'eucl', 'timetype', 'frame') % frames as x axis unit
mcplottimeseries(s, [12 14], 'var', 'quat', 'dim', 2, 'plotopt', 'comb') % all in one plot, component 2

**comments**

**see also**
mcpotenergy

**synopsis**
Estimates the instantaneous potential energy of each body segment.

**syntax**
\[
\text{pe} = \text{mcpotenergy}(d, \text{segd}, \text{segmpar})
\]

**input parameters**
- \(d\): MoCap data structure
- \(\text{segd}\): segm data structure calculated from \(d\)
- \(\text{segmpar}\): segmpar structure (see \text{mcgetsegmpar}\)

**output**
- \(\text{pe}\) = matrix containing potential energy values for each body segment

**examples**
\[
\begin{align*}
\text{segd} &= \text{mcj2s}(d, \text{j2spar}); \\
\text{spar} &= \text{mcgetsegmpar}('\text{Dempster}', \text{segmnum}); \\
\text{pe} &= \text{mckinenergy}(d, \text{segd}, \text{spar});
\end{align*}
\]

**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**
- \text{mcj2s}, \text{mcgetsegmpar}, \text{mckinenergy}
mcread

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

**syntax**
```
d = mcread(fn);
d = mcread;
```

**input parameters**
- `fn`: file name, tsv, c3d, bvh, 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 = mcread('filename.tsv');
d = mcread('filename.c3d');
d = mcread('filename.bvh');
d = mcread('filename.mat');
d = mcread('filename.wii');
d = mcread;
```

**comments**
Currently the .c3d, .tsv (as exported by QTM), .bvh, .mat (as exported by QTM), and .wii (WiiDataCapture software) formats are supported. The file names must have postfixes '.c3d', '.tsv', '.bvh', '.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.

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. If further problems occur when reading in .c3d files, try to adapt the ‘machinetype' parameters as indicated in the readc3d.m (in the folder ‘private').

Reading in .bvh files requires additional toolboxes available here: http://staffwww.dcs.shef.ac.uk/people/N.Lawrence/mocap/ (mocap and ndlutil).

**see also**
mcreademg

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

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

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

**output**
- d: norm data structure

**examples**
- \[d = mcreademg('filename.tsv');\]

**comments**

**see also**
- mcfilteremg
mcreorderdims

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

**syntax**  
\[
d2 = \text{mcreorderdims}(d, \text{dims});
\]

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

**output**  
- d2: MoCap data structure

**examples**  
\[
d2 = \text{mcreorderdims}(d, [1 3 2]);
\]

**comments**

**see also**
mcrepovizz

**synopsis**
Exports MoCap structure as valid repoVizz .csv files, .xml repoVizz struct and optional .bones file.

**syntax**
mcrepovizz(d, path, p);

**input parameters**
d: MoCap data structure
path: path to save the .csv files (optional). If no path is given, files are saved in the current directory. If the chosen directory does not exist, it will be created.
p: animpar parameter structure (optional)

**output**
.csv files and repoVizz struct .xml file saved in the current or in the specified directory. Optional .bones file saved as well if animpar structure specified in the third argument

**examples**
mcrepovizz(d)
mcrepovizz(d, 'folder')
mcrepovizz(d, '/path/folder') % (only Mac)
mcrepovizz(d, 'folder', p)
mcrepovizz(d, p)

**comments**
For info about repoVizz: http://repovizz.upf.edu/

**see also**
mcread, mcwritetsv
mcresample

**synopsis**
Resamples motion capture data using interpolation.

**syntax**
```
d2 = mcresample(d, newfreq, method);
```

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

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

**examples**
```
d2 = mcresample(d, 240);
d2 = mcresample(d, 360, 'spline');
```

**comments**

**see also**
mcreverse

**synopsis**
Reverses dimensions of motion-capture data.

**syntax**
\[ d2 = mcreverse(d, x); \]

**input parameters**
- \( d \): MoCap structure or data matrix
- \( x \): reverse vector (set 1 for each dimension to be reversed, otherwise 0)

**output**
- \( d2 \): MoCap structure or data matrix

**examples**
\[ d2 = mcreverse(d, [0 0 1]); \]

**comments**

**see also**
mcrotate

**synopsis**
Rotates motion-capture data.

**syntax**

\[
d2 = \text{mcrotate}(d, \text{theta});
\]

\[
d2 = \text{mcrotate}(d, \text{theta}, \text{axis});
\]

\[
d2 = \text{mcrotate}(d, \text{theta}, \text{point});
\]

\[
d2 = \text{mcrotate}(d, \text{theta}, \text{axis}, \text{point});
\]

**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**

\[
d2 = \text{mcrotate}(d, 130); \% \text{ rotate 130 degrees counterclockwise around the vertical axis}
\]

\[
d2 = \text{mcrotate}(d, 90, [1 0 0]); \% \text{ rotate around the x axis}
\]

\[
d2 = \text{mcrotate}(d, 45, [0 1 0], [0 0 500]); \% \text{ rotate around the axis parallel to y axis going through point [0 0 500]}
\]

\[
d2 = \text{mcrotate}(d, 20, [], [0 1000 0]); \% \text{ rotate around the z (vertical) axis going through point [0 1000 0]}
\]

**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
**mcrotationrange**

**synopsis**
Calculates the rotation range between two markers.

**syntax**
\[
f = \text{mcrotationrange}(d, m1, m2);
\]

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

**output**
- \(r\): rotation range (the higher the value, the more rotation)

**examples**
\[
r = \text{mcrotation}(d, 13, 17);
\]

**comments**

**see also**

**references**
mcs2j

**synopsis**
Performs 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**
```matlab
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
\[ ds = \text{mcsethares}(d, \text{per}); \] %small-to-large Sethares transform
\[ [ds, \text{pers}, \text{pows}] = \text{mcsethares}(d, \text{per}, \text{nbasis}); \] %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/downloader.html

see also
- mcpcaproy, mcicaproy
mcsetlength

synopsis
Sets mocap data to the length given.

syntax
\[ d2 = mcsetlength(d, n) \]
\[ d2 = mcsetlength(d, n, 'timetype', 'sec') \]
\[ d2 = mcsetlength(d, n, 'position', 'location') \]

input parameters
- d: MoCap or norm data structure
- n: new length of mocap data
- timetype: length given in frames ('frame') or seconds ('sec') (default: sec)
- location: position where to add or trim frames ('beginning' or 'end' – default: end)

output
- d2: MoCap or norm data structure

examples
\[ d2 = mcsetlength(d, 1200); \]
\[ d2 = mcsetlength(d, n, 'timetype', 'sec'); \]
\[ d2 = mcsetlength(d, 1200, 'location', 'beginning'); \]

comments
If the given length is less than the number of frames in the mocap data, the data will be trimmed to the given length from either beginning or end. If the given length is more than the number of frames in the mocap data, data will be added by replicating with first or last frame.

see also
mcaddframes, mctrim
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). The data set must have the same amount of markers as indicated in mnum.
mnum: vector containing the marker numbers to be replaced in the original data set (order as in replacement mocap structure)

**output**
d2: MoCap structure

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

**comments**
Use mcgetmarker to shorten the replacing data set to fit the mnum vector.
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. Empty marker names will be set to EMPTY and can be adapted manually if desired.

**see also**
mccombine, mcgetmarker
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
\[
m = \text{mcskewness}(d);
\]

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
\(\text{mcmean}, \text{mcstd}, \text{mcvar}, \text{mckurtosis}\)
mcsmoothen

synopsis
Smoothen 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
- \(\text{filterparams}\): order and cutoff frequency for Butterworth filter (optional, default \([2, 0.2]\))
- \(\text{method}\): Butterworth filtering is default - if Savitzky-Golay filtering is to be used, use 'acc' as method argument
- \(\text{window}\): 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
\[
d2 = \text{mcsmoothen}(d); \quad \% \text{Butterworth filter smoothing with default parameters}
\]
\[
d2 = \text{mcsmoothen}(d, [2 \ 0.1]); \quad \% \text{second order Butterworth filter with 0.1 Hz cutoff frequency}
\]
\[
d2 = \text{mcsmoothen}(d, 'acc'); \quad \% \text{S-G filter smoothing with default frame length}
\]
\[
d2 = \text{mcsmoothen}(d, 9); \quad \% \text{S-G filter smoothing using a 9-frame window}
\]

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 \texttt{sgolayfilt}.

see also
- \texttt{mctimeder}
**mcsort**

**synopsis**
sorts mocap data according to marker names (alphanumerical or according to given numeric or cell array indicating marker numbers or markers names as to how the output data is to be sorted).

**syntax**
- `d2=mcsort(d)`
- `d2=mcsort(d, srt)`

**input parameters**
- **d**: MoCap data structure
- **srt**: numeric or cell array containing markers numbers or marker names (optional)

**output**
- **d2**: reordered mocap data structure

**examples**
- `d2=mcsort(d)`
- `d2=mcsort(d, [1:5 7 6 9 8 10:20]);`
- `d2=mcsort(d, d1.markerName);`

**comments**
If the sort variable is not given, the data will be sorted alphanumerical according to the marker names.
The number of items in the sort array has to match the number of markers in the input mocap data structure.

**see also**
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: 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**

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

**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**

\[
d2 = \text{mctimeder}(d); \ % \ \text{first-order time derivative using the fast method (Butterworth filter with default parameters)}
d2 = \text{mctimeder}(d, [2 .1]); \ % \ \text{first-order time derivative using fast version (second order Butterworth filter with 0.1 Hz cutoff frequency)}
d2 = \text{mctimeder}(d, 'acc'); \ % \ \text{first-order time derivative using the accurate version (Savitzky-Golay filter)}
d2 = \text{mctimeder}(d, 2, 9, 'acc'); \ % \ \text{second-order time derivative with 9-frame window using the accurate version (Savitzky-Golay filter)}
\]
**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 $4*n+w-4$. 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**
\[
d2 = mctimeintegr(d);
d2 = mctimeintegr(d, order);
\]

**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**
\[
d2 = mctimeintegr(d, 2); \quad \text{% second-order time integral}
\]

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

**see also**
mctimedener
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**
**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
- `timetype`: either 'sec' (default) or 'frame'

**output**
- `d2`: MoCap, norm, or segm data structure containing frames from `t1` to `t2` (if `timetype == 'frame'`) or frames between `t1` and `t2` seconds (if `timetype == 'sec'`) of MoCap data structure `d`.

**examples**
- `d2 = \text{mctrim}(d, 305, 1506, 'frame');`
- `d2 = \text{mctrim}(d, 3, 5, 'sec');`

**comments**

**see also**
mcvar

synopsis
Calculates the variance of data, ignoring missing values.

syntax
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
Converting MoCap structure vector to 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
```
memean, mcstd, mcvar, mcskewness, mckurtosis, mcperiod
```
mcwritetsv

**synopsis**
Saves mocap structure as a tsv file.

**syntax**
mcwritetsv(d, path)

**input parameters**
d: MoCap data structure  
path: path to save the tsv file (optional). If no path is given, file is saved to current directory

**output**
tsiv file, saved in the current or in the specified directory

**examples**
mcwritetsv(d)
mcwritetsv(d, 'folder')  
mcwritetsv(d, '/path/folder') % (Mac)

**comments**

**see also**
mcread
Never confuse motion with action.

— Benjamin Franklin —