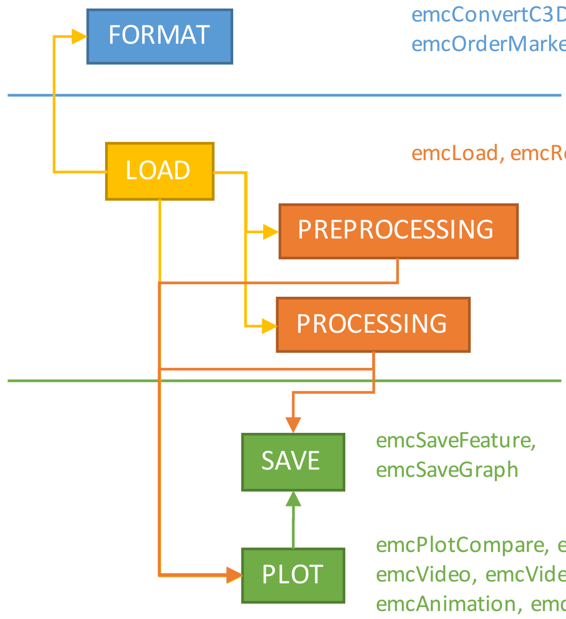
# Expressive Motion Capture TOOLBOX

## SCHEMA TOOLBOX



emcConvertC3D2TSV, emc DivideMultipleBodies, emc MidpointBaricenter, emcOrderMarker, emcVerticalLine, emcAlignHeightMarkerBaricenter

emcLoad, emcRemovePrefix, emcClassification, emcAbsolutePosition

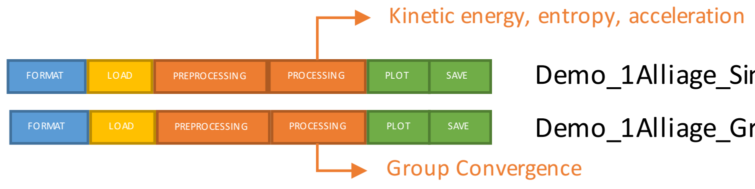
emcPreprocessing

emcDirectness, emcDistance, emcEventSync, emcFluidity, emcGroupConvergence, emcHandsSpeed.m emcHeatmap,emcHeatmapData, emcKineticEnergy, emcMCKineticEnergy, emcMCPotentialEnergy,emcPositionHands, emcPotentialEnergy, emcSampleEntropyKinEnergy, emcSymmetry, emcUpwardMovement, emcAngleVector, emcConvexhull, emcCumulativeDistance, emcDensity, emcDerivative,

emcSaveFeature, emcSaveGraph

emcPlotCompare, emcPlotMarker, emcPlotQuiver, emcVideo, emcVideoGroup, emcVisualSequence, emcAnimation, emcPlotBody, emcPlotBody2Dn emcPlotBody3D

## PROJECTS



ALLIAGE

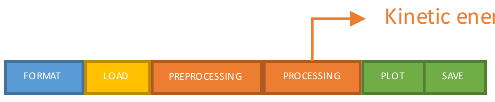
| | | |
|---|---|---|
| single | Kinetic energy, entropy, acceleration | Demo_1Alliage_Single.m |
| group | Group Convergence | Demo_1Alliage_Group.m |

EMODANCE

FORMAT
LOAD
PREPROCESSING
PROCESSING
PLOT
SAVE

Demo_2Emodance.m

MocoV3



Kinetic energy, angles

Demo_3MocoV3.m

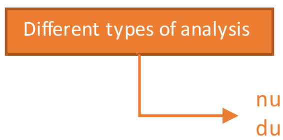
IMPROV

- (Not yet ready with the new toolbox)

ORCHESTRA

- (Not yet ready with the new toolbox)

JPO



Demo_4JPO.m

numberplot, deplacementhistogram, nbroomhistogram, durationparthistogram, durationroomhistogram

S. Schaerlaeken - NEAD, University of Geneva

## PRIMER: LOAD – PREPROCESSING – PROCESSING – PLOT (PART 1)

```
%% PATH
pathComputer = '/Users';

%% TOOLBOX
% Add path for toolbox
addpath(genpath([pathComputer, filesep, '0434_EMC_Toolbox']))
```

```
%% LOAD
% Load a file - 1 Body
cfg = []; % Empty cfg
cfg.fillGapFlag = true;
cfg.removePrefix = {'S01_'};
cfg.absolutePositionFlag = true;
cfg.headMarkers = {'LFHD','RFHD','OZ'};
% Classification
cfg.classificationFlag = true;
cfg.classificationType = 'prefix';
cfg.classInfo = {'whole_anticipation','piano_forte'};
cfg.classPrefixPosition = {1,3}; % Give the range where the prefix is
positioned within the filename
cfg.classListPrefix = {{'w','a'},...
                   {'p','f'}}; % List of possible prefixes
cfg.classListValue = {{1,2},...
                  {1,2}}; % Categorisation number
tsvFile1 = emcLoad([pathComputer, filesep, 'example1.tsv'],cfg);
tsvFile2 = emcLoad([pathComputer, filesep, 'example2.tsv '],cfg);
```

```
%% PLOT
% Body
cfg.connectionMatrix = [1 2;2 3;1 3;3 4;4 5;4 6;4 13;5 7;7 9;6 8;8 10; 11 12; 4 14];
emcPlotBody(tsvFile1, cfg)
emcPlotBody(tsvFile2, cfg)
% Quiver
cfg.plotBodyFlag = true;
emcPlotQuiver(tsvFile1, cfg)
emcPlotQuiver(tsvFile2, cfg)

% Markers
cfg.plotMarkerList = {'LWRB', 'RWRB'};
emcPlotMarker(tsvFile1, cfg)
% Compare
cfg.plotMarkerList = {'RWRB'};
emcPlotCompare(tsvFile1, tsvFile2, cfg)
```

```
%% PREPROCESSING
cfg.preprocessing.preprocessingType = {'center','rotation frontal'};
cfg.preprocessing.markerFrontalList = {'LSHO','RSHO'};
tsvFile1 = emcPreprocessing( tsvFile1,cfg.preprocessing);
tsvFile2 = emcPreprocessing( tsvFile2,cfg.preprocessing);
% Plot to see the difference
emcPlotBody(tsvFile1, cfg)
```

## PRIMER: LOAD – PREPROCESSING – PROCESSING – PLOT (PART 2)

```
%% PROCESSING
% Display
cfg.display = true;
% Speed
cfg.deriv = 1;
cfg.euclidianFlag = true;
tsvFile1 = emcDerivative(tsvFile1,cfg);
tsvFile2 = emcDerivative(tsvFile2,cfg);
% Convexhull
tsvFile1 = emcConvexhull(tsvFile1,cfg);
tsvFile2 = emcConvexhull(tsvFile2,cfg);
% Synchronisation
cfg.syncType = 'intra';
cfg.dataType = 'marker';
cfg.dataList = {'LELB';'RELB';'LWRB';'RWRB'};
[ syncQ, syncq ] = emcEventSync( tsvFile1, cfg );
[ syncQ, syncq ] = emcEventSync( tsvFile2, cfg );
```

```
%% MORE PLOT
cfg.step = 30;
cfg.width = 400;
cfg.visSeqFeature = 'speed';
emcVisualSequence(tsvFile1, cfg)

%% SAVE
emcSaveGraph([pathComputer, filesep,'Output'])
```

## FORMAT

### emcConvertC3D2TSV

<u>description</u>
Opens a C3D file and convert it into a TSV file. TSV file is saved afterwards

<u>syntax</u>
emcConvertC3D2TSV( filename, cfg );

<u>input parameters</u>
filename: string containing the path or filename of the file to convert
cfg: configuration structure
  [MANDATORY]
  -
  [OPTIONAL]
  *.outputDir: string containing the path to the output directory
  *.deleteMarker: cell array of markers names to be deleted
  *.changeMarker: cell array of markers names to be changed
  *.orderMarker: cell array containing the final order of the list of markers names
  *.fillgapFlag: boolean indicating if use fillgap or not (from Motion Capture Toolbox)

<u>output</u>
[]

<u>Examples</u>
```
cfg.outputDir = 'C:\Users\Desktop';
cfg.deleteMarker  = {'d'};
cfg.changeMarker = {'tutu','e';  'baba','f'};
cfg.orderMarker = {'a','b','e','f','c'};
cfg.fillgapFlag = true;
emcConvertC3D2TSV( 'motion.c3d', cfg );
```

<u>comments</u>
-

<u>see also</u>
-

## emcDivideMultipleBodies

description
Divides a TSV file containing multiple bodies (represented by prefixe) into multiple files containing one body and save them

syntax
emcDivideMultipleBodies(filename, cfg)

input parameters
filename: string containing the path or filename of the file to convert
cfg: configuration structure
  [MANDATORY]
  *.prefixBodyList: cell array of prefix name for every body
  [OPTIONAL]
  *.outputDir: string containing the path to the output directory

output
[]

Examples
```
cfg.prefixBodyList = {'body1_','body2_'};
cfg.outputDir = 'C:\Users\Desktop';
emcDivideMultipleBodies('motion.c3d', cfg );
```

comments
Saves the different file as "filename_prefixBody.tsv"

see also
-

## emcMidpointBaricenter

description
Computes the baricenter of a list of multiple markers and the middle point between the first two markers of the list. The baricenter and middle point are added to the marker list in the tsvFile

syntax
tsvFile = emcMidpointBaricenter(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.transformMarker: cell array of markernames on which the baricenter and the midpoint will be computed
*.transformName: string containing the name of the group of markers creating the baricenter and middle point
[OPTIONAL]
-

output
tsvFile: MoCap data structure

Examples
```
cfg.transformMarker = {'a','b','c'};
cfg.transformName = {'abc'};
tsvFile = emcMidpointBaricenter(tsvFile, cfg);
```

comments
The markers created are a combination of the transformName and either "Bar" for Baricenter or "Mid" for Middle point

see also
emcVerticalLine

## FORMAT

### emcOrderMarker

<u>description</u>
Reorders tsvFile placing markers into a specific order

<u>syntax</u>
tsvFileOrdered = emcOrderMarker( tsvFile, cfg );

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.orderMarker: cell array of markernames in the specific order
  [OPTIONAL]
  -

<u>output</u>
tsvFileOrder: MoCap data structure (with marker in a different order)

<u>examples</u>
cfg.orderMarker = {'b','a','c'};
tsvFile = emcOrderMarker( tsvFile, cfg );

<u>comments</u>
-

<u>see also</u>
-

## emcVerticalLine

<u>description</u>
Creates a second marker right above a specific marker in order to draw a vertical line between them

<u>syntax</u>
tsvFile = emcVerticalLine( tsvFile, cfg );

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.transformMarker: cell array or string containing markername(s)
  [OPTIONAL]
  -

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.transformMarker = {'a'};
tsvFile = emcMidpointBaricenter(tsvFile, cfg);
```

<u>comments</u>
The markers created are a combination of the markername and "Vert" for Vertical line.
This can be further use to compare angle to a straight line or to have an idea of how bended the movement is compared to a straight vertical line

<u>see also</u>
emcMidpointBaricenter

## FORMAT

### emcAlignHeightMarkerBaricenter

<u>description</u>
Aligns markers to the height of the baricenter of a collection of markers.

<u>syntax</u>
tsvFile = emcAlignHeightMarkerBaricenter(tsvFile, cfg);

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.transformMarker: string containing the path to the output directory
  *.alignMarker: cell array of markers names to be deleted
  [OPTIONAL]
  -

<u>output</u>
[]

<u>examples</u>
```
cfg.transformMarker = {'a','b'};
cfg.alignMarker = {'c'};
tsvFile = emcAlignHeightMarkerBaricenter(tsvFile, cfg);
```

<u>comments</u>
-

<u>see also</u>
-

## emcLoadSingle

description
Loads a single tsv file from the filename into a structure

syntax
```
tsvFile = emcLoadSingle(filename, cfg);
```

input parameters
filename: string containing the path or filename of the file to load
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.makersToKeep: cell array containing the markers names to be kept in the tsv structure
*.fillgapFlag: boolean indicating if use fillgap or not (from Motion Capture Toolbox)
*.classificationFlag: boolean indicating if emcClassification should be run
*.absolutePositionFlag: boolean indicating if emcAbsolutePosition should be run
*.removePrefixFlag: boolean indicating if emcRemovePrefix should be run

output
tsvFile: MoCap data structure

examples
```
tsvFile1 = emcLoadSingle('motion1.tsv', cfg);

cfg.makersToKeep = {'a','b','c'};
cfg.fillgapFlag = true;
cfg.absolutePositionFlag = true;
cfg.removePrefixFlag = true;
cfg.removePrefix = {'body1_'};
cfg.classificationFlag = true;
cfg.classificationType = 'prefix';
cfg.classInfo = {'className1','className2'};
cfg.classPrefixPosition = {1,2};
cfg.classListPrefix = {{'1','2','3'},...
                 {'1','2'}};
cfg.classListValue = {{'class1_1','class1_2','class1_3'},...
                {'class1_2','class2_2'}};
tsvFile2 = emcLoadSingle('motion2.tsv', cfg);
```

comments
-

see also
emcClassification
emcAbsolutePosition
emcRemovePrefixFlag

## emcClassification

description
Adds tags to the tsv structure for classification. This can be made according to different methods: "prefix", uses the filename given and extract information at specific location

syntax
tsvFile = emcClassification(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.classificationType: define the classification method
  if cfg.classificationType: 'prefix'
    *.classInfo: cell array containing the classes name
    *.classPrefixPosition: cell array containing the indexes of the classification letters in the filename
    *.classListPrefix: cell array containing the different options available at the position mentionned in cfg.classPrefixPosition
    *.classListValue: cell array containing the value of each of the options available in cfg.classListPrefix
  [OPTIONAL]
  -

output
tsvFile: MoCap data structure

examples
```
cfg.classificationType = 'prefix';
cfg.classInfo = {'className1','className2'};
cfg.classPrefixPosition = {1,2};
cfg.classListPrefix = {{'1','2','3'},...
                       {'1','2'}};
cfg.classListValue = {{'class1_1','class1_2','class1_3'},...
                      {'class1_2','class2_2'}};
tsvFile = emcClassification(tsvFile, cfg);
```

comments
if store the classification in tsvFile.info.classification

see also
emcLoadSingle

## LOAD

### emcAbsolutePosition

description
Store the absolute position of tsvFile (timecourse of the first marker).
Useful if need to be modify with for example preprocessing steps such as mccenter.

syntax
```
tsvFile = emcAbsolutePosition(tsvFile, cfg);
```

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  -
  [OPTIONAL]
  *.headMarkers: cell array with markers of the head

output
tsvFile: MoCap data structure

examples
```
cfg.headMarkers = {'head1','head2'};
tsvFile = emcAbsolutePosition(tsvFile, cfg);
```

comments
Stores the absolute postion in tsvFile.info.absolutePosition

see also
emcLoadSingle

## emcRemovePrefix

description
Removes a prefix / different prefixes to the list of markers of a tsvFile

syntax
```
tsvFile = emcRemovePrefix(tsvFile, cfg);
```

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
    [MANDATORY]
    *.removePrefix: string or cell array containing the prefix
    [OPTIONAL]
    -

output
tsvFile: MoCap data structure

examples
```
cfg.removePrefix = {'body1_'};
tsvFile = emcRemovePrefix(tsvFile, cfg);
```

comments
stores the prefix in tsvFile.info.removePrefix

see also
emcLoadSingle

# PREPROCESSING

## emcPreprocessing

description
Preprocessing of a tsvFile. Includes methods such as mc2frontal, mcrotate, mccenter, mccenter_marker, mcnorm, and mcsmoothen
from the Motion Capture Toolbox, Copyright 2008, University of Jyvaskyla, Finland

syntax
tsvFile = emcPreprocessing(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.preprocessingType: cell array with the type(s) of preprocessing
   method to be applied
  if cfg.preprocessingType: 'rotation frontal'
    *.markerFrontalList: list with the markers index defining the frontal plane (see mc2frontal)
  if cfg.preprocessingType: 'rotation'
    *.rotationAngleValue: value of the rotation (see mcrotate)
    *.rotationAngleAxis: list defining the axis (see mcrotate)
  if cfg.preprocessingType: 'center marker'
    *.markerToCenter: string marker name of the marker on which the rest of the data is
centered
  if cfg.preprocessingType: 'smoothen'
    *.smoothValue: int smooth value (see mcsmoothen)
  [OPTIONAL]
  -

output
tsvFile: MoCap data structure

examples
cfg.preprocessingType = {'rotation frontal', 'rotation', 'center', 'center marker', 'normalize', 'smoothen'};
cfg.markerFrontalList = [1 2];
cfg.rotationAngleValue = 30;
cfg.rotationAngleAxis = [1 0 0];
cfg.markerToCenter = 'head';
cfg.smoothValue = 20;
tsvFile = emcPreprocessing(tsvFile, cfg);

comments

see also
mc2frontal
mcrotate
mccenter
mccenter_marker
mcnorm
mcsmoothen

## emcConvexhull

<u>description</u>
Computes the convexhull

<u>syntax</u>
tsvFile = emcConvexhull(tsvFile, cfg)

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.frame: int value of the frame at which the body should be displayed
*.dispay: boolean deciding if a figure is to be plotted (default: true)
if cfg.display: true
*.connectionMatrix: [mandatory] list of indexes defining the connection of the body (see emcPlotBody3D)

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.featMarker   = {'a','b','c'};
cfg.frame  =  40;
cfg.display = true
cfg.connectionMatrix = [1 2; 2 3; 1 3]
tsvFile = emcConvexhull(tsvFile, cfg);
```

<u>comments</u>
feature is saved in tsvFile.processing.convexhull

<u>see also</u>
-

## PROCESSING

### emcCumulativeDistance

description
Computes the cumulative distance

syntax
tsvFile = emcCumulativeDistance(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featMarker   = {'a','b','c'};
cfg.display = true
tsvFile = emcCumulativeDistance(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.cumuldist

see also
-

## PROCESSING

### emcDensity

description
Computes the density - the average distance of all markers from the baricenter

syntax
tsvFile = emcDensity(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
    -
  [OPTIONAL]
  *.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
  *.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featMarker  =  {'a','b','c'};
cfg.display = true
tsvFile = emcDensity(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.density

see also
-

## PROCESSING

### emcDerivative

<u>description</u>
Computes the derivative of the motion

<u>syntax</u>
tsvFile = emcDerivative(tsvFile, cfg)

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.deriv: int order of the derivative to be calculated - 1 (speed), 2 (acceleration), 3 (jerk)
*.euclidianFlag: boolean to determine if the script uses euclidian norm after derivative
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.featMarker   =  {'a','b','c'};
cfg.display = true
tsvFile = emcDerivative(tsvFile, cfg);
```

<u>comments</u>
feature is saved in tsvFile.processing.speed
feature is saved in tsvFile.processing.acceleration
feature is saved in tsvFile.processing.jerk

<u>see also</u>
-

# PROCESSING

## emcDirectness

description
Computes the directness - Movement Directness Index is computed from a trajectory drawn in the space by a joint as the ratio between the eu-clidean distance, calculated between the starting and the ending point of the trajectory, and the trajectory?s actual length. [Piana, 2013]

syntax
tsvFile = emcDirectness(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featMarker  = {'a','b','c'};
cfg.display = true
tsvFile = emcDirectness(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.directness

see also
-

# PROCESSING

## emcDistance

description
Computes the distance between 2 markers or to the ground from one marker

syntax
tsvFile = emcDistance(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.featDistMarker: cell array containing 2 markernames or 1 markernames and "ground"
[OPTIONAL]
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featDistMarker = {'a','b'};
cfg.display = true
tsvFile = emcDistance(tsvFile, cfg);

cfg.featDistMarker = {'a','ground'};
cfg.display = true
tsvFile = emcDistance(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.distance

see also
-

## emcEventSync

description
Computes the synchronicity matrix for different markers, bodies or features as defined by Albordo, 2016

syntax
tsvFile = emcEventSync(tsvFile1[, tsvFile2], cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.syncType: str type of synchrony - 'intra' (for one body, compares inside one body), 'inter' (for multiple body, compares bodies one against the other)
*.dataType: str type of data - 'marker' (on the marker motion), 'feature' (on the computed features)
[OPTIONAL]
*.dataList: cell array containing the markernames or feature names on which the sync is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
syncQ: table overall degree of sync
syncq: table overall degree of delay

examples
```
cfg.syncType = 'intra';
cfg.dataType = 'marker';
cfg.dataList = {'a','b','c'};
cfg.display = true
tsvFile = emcEventSync(tsvFile, cfg);

cfg.syncType = 'inter';
cfg.dataType = 'feature';
cfg.dataList = {'speed', 'convexhull'};
cfg.display = true
tsvFile = emcEventSync(tsvFile1, tsvFile2, cfg);
```

comments
feature is saved in tsvFile.processing.directness

see also
-

## PROCESSING

### emcFluidity

description
Computes the fluidity
The principle of the curvature can be applied to the movement velocity and its variation in time to calculate the movement fluidity (FI): high curvature of the speed trajectory in timemeans low fluidity, while low curvaturemeans high fluidity. To calculate the fluidity we compute the curvature of the tangential velocity of the desired joint. [Piana, 2013]

syntax
tsvFile = emcFluidity(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featMarker  = {'a','b','c'};
cfg.display = true
tsvFile = emcFluidity(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.fluidity

see also
-

## PROCESSING

### emcHandsSpeed

<u>description</u>
Computes the speed of the hands

<u>syntax</u>
tsvFile = emcHandsSpeed(tsvFile, cfg)

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.handsMarkers: cell array containing the markernames for the hands
*.euclidianFlag: boolean to determine if the script uses euclidian norm after derivative
[OPTIONAL]
*.dispay: boolean deciding if a figure is to be plotted (default: true)

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.handsMarkers = {'a','b'};
cfg.display = true
cfg.euclidianFlag = true;
tsvFile = emcHandsSpeed(tsvFile, cfg);
```

<u>comments</u>
feature is saved in tsvFile.processing.speed

<u>see also</u>
-

## PROCESSING

### emcSymmetry

<u>description</u>
Computes the symmetry between the left and right part of the body as defined by Piana 2013

<u>syntax</u>
tsvFile = emcSymmetry(tsvFile, cfg)

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.leftPrefix: string prefix of the left side of the body markernames
-- they should all start with this prefix
*.rightPrefix: string prefix of the right side of the body markernames
-- they should all start with this prefix
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.leftPrefix = 'L';
cfg.rightPrefix = 'R';
cfg.featMarker = {'Ra','Rb','Lc','Ld'};
cfg.display = true
tsvFile = emcSymmetry(tsvFile, cfg);
```

<u>comments</u>
feature is saved in tsvFile.processing.symmetry

<u>see also</u>
-

## PROCESSING

### emcUpwardMovement

description
Computes the amount of upward movement, everytime a marker is going up on the Z axis, the value for the feature increases

syntax
tsvFile = emcUpwardMovement(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.featMarker: cell array containing the markernames on which the feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featMarker  =  {'a','b','c'};
cfg.display = true
tsvFile = emcUpwardMovement(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.upwardMovement

see also
-

## PROCESSING

### emcHeatmap

description
Computes the heatmap of the baricenter motion seen only on a 2D plane (x-y) from above.

syntax
tsvFile = emcHeatmap(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.heatmapFeature: str feature name to modulate the value of the heatmap

output
-

examples
```
cfg.heatmapFeature = 'speed';
cfg.reduceFlag = true;
cfg.reduceResolution = [2,3];
emcHeatmap(tsvFile, cfg);
```

comments
-

see also
emcHeatmapData

## emcHeatmapData

description
Computes the heatmap of the baricenter motion seen only on a 2D plane (x-y) from above.

syntax
tsvFile = emcHeatmap(dataMotion[, dataFeature], cfg)

input parameters
dataMotion: table motion data (3 columns per marker, each line represent the position of the marker at time t)
dataFeature: table feature data
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.reduceFlag: boolean flag meaning that the final resolution will be reduced
if reduceFlag: true
*.reduceResolution: [mandatory] table with 2 values - the reduction factor for X and Y.

output
-

examples
```
cfg.reduceFlag = true;
cfg.reduceResolution = [2,3];
emcHeatmapData(motionData, featureData, cfg);
```

comments
-

see also
emcHeatmap

## emcGroupConvergence

<u>description</u>
Computes convergence of lines of sight. Either globally between a fixed point and the intersection of all lines for sight or partially between the different bodies

<u>syntax</u>
tsvFile = emcGroupConvergence(tsvFile,tsvFile2[, tsvFile3,...], cfg)

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.convergenceType: str type of convergence - global: calculating the distance between the intersection of all line of sight and a fixed point. Partial: calculating the distance between one body and the line of sight converging of all the other bodies
*.headMarker: cell array containing the markernames for the head defininf the line of sight (suggestion: use baricenter and middlepoint)
[OPTIONAL]
*.dispay: boolean deciding if a figure is to be plotted (default: true)
*.displayVerification: boolean deciding if a figure is to be plotted for verification of the line of sights(default: false)

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.convergenceType = 'global';
cfg.headMarker = {'HEADBar','HEADMid'}
tsvFile = emcGroupConvergence(tsvFile,tsvFile2, tsvFile3, cfg));
```

<u>comments</u>
feature is saved in tsvFile.processing.groupConvGlobal/tsvFile.processing.groupConvPartial

<u>see also</u>
-

## emcAngleVector

description
Computes the angle between 2 lines, each of which is defined by 2 markers

syntax
tsvFile = emcAngleVector(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.angleName: str name of the angle
*.angleVector: cell array defining the 2 lines (4 markers)
[OPTIONAL]
*.angleDim: int value representing the dimension in which the angle is calculated -- if 2: angle is computed only in the X-Y space. if 3: angle is computed in the X-Y-Z space (default: 3)
feature is calculated (default: all)
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.angleName = 'bending';
cfg.angleVector = {{'a','b'},{'c','d'}};
cfg.angleDim = 3;
cfg.display = true;
tsvFile = emcAngleVector(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.(angleName)

see also
-

## emcKineticEnergy

description
Computes the kinetic energy of the markers (based on mcpotenergy from the Motion Capture Toolbox, Copyright 2008, University of Jyvaskyla, Finland
)

syntax
tsvFile = emcKineticEnergy(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.m2jpar: vector contraining the index of the markers for the transformation to joints (see the Motion Capture Toolbox)
*.j2spar: vector contraining the index of the joints for the transformation to segments (see the Motion Capture Toolbox)
*.mcspar: vector contraining the weights of all the segments (see the Motion Capture Toolbox)
*.paramComputation: str flag defining the mean of concatenation of the data - 'mean','sum','median'
[OPTIONAL]
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.m2jpar.type = 'm2jpar';
cfg.m2jpar.nMarkers = 22;
cfg.m2jpar.markerNum = {[4,9,11,12],[4,11],[4,5,1],[1,2,3]};
cfg.m2jpar.markerName = {'PELVIS', 'RPELV', 'RTHIGH', 'RTIB'};
cfg.j2spar.type = 'j2spar';
cfg.j2spar.rootMarker = 1;
cfg.j2spar.frontalPlane = [1 2];
cfg.j2spar.parent = [0 1 2 3];
cfg.j2spar.segmentName = cfg.m2jpar.markerName;
cfg.segmidx = [0 1 8 7];
cfg.mcspar = mcgetsegmpar('Dempster', cfg.segmidx);
cfg.paramComputation = 'mean';
cfg.display = true;
tsvFile = emcKineticEnergy(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.kinEnerg

see also
mckinenergy

## PROCESSING

### emcPotentialEnergy

description
Computes the potential energy of the markers (based on mcpotenergy from the Motion Capture Toolbox, Copyright 2008, University of Jyvaskyla, Finland )

syntax
tsvFile = emcPotentialEnergy(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.m2jpar: vector contraining the index of the markers for the transformation to joints (see the Motion Capture Toolbox)
  *.j2spar: vector contraining the index of the joints for the transformation to segments (see the Motion Capture Toolbox)
  *.mcspar: vector contraining the weights of all the segments (see the Motion Capture Toolbox)
  *.paramComputation: str flag defining the mean of concatenation of the data - 'mean','sum','median'
  [OPTIONAL]
  *.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.m2jpar.type = 'm2jpar';
cfg.m2jpar.nMarkers = 22;
cfg.m2jpar.markerNum = {[4,9,11,12],[4,11],[4,5,1],[1,2,3]};
cfg.m2jpar.markerName = {'PELVIS', 'RPELV', 'RTHIGH', 'RTIB'};
cfg.j2spar.type = 'j2spar';
cfg.j2spar.rootMarker = 1;
cfg.j2spar.frontalPlane = [1 2];
cfg.j2spar.parent = [0 1 2 3];
cfg.j2spar.segmentName = cfg.m2jpar.markerName;
cfg.segmidx = [0 1 8 7];
cfg.mcspar = mcgetsegmpar('Dempster', cfg.segmidx);
cfg.paramComputation = 'mean';
cfg.display = true;
tsvFile = emcPotentialEnergy(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.potEnerg

see also
mcpotenergy

## emcPositionHands

<u>description</u>
Computes the amount of time the hands spend in either the upper, middle or lower part of the body. The upper part corresponds to "above the head", the middle corresponds to "between the head and pelvis", lower part corresponds to "below the pelvis"

<u>syntax</u>
tsvFile = emcPositionHands(tsvFile, cfg)

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.headMarker: cell array containing one markername for the head
*.pelvisMarker: cell array containing one markername for the pelvis
*.handsMarker: cell array containing two markername for the two hands
[OPTIONAL]
*.dispay: boolean deciding if a figure is to be plotted (default: true)

<u>output</u>
tsvFile: MoCap data structure

<u>examples</u>
```
cfg.headMarker = {'a'};
cfg.pelvisMarker = {'a'};
cfg.handsMarker = {'c','d'};
cfg.display = true;
tsvFile = emcPositionHands(tsvFile, cfg);
```

<u>comments</u>
feature is saved in tsvFile.processing.positionHands

<u>see also</u>
-

## PROCESSING

### emcSampleEntropy

description
Computes the sample entropy of a feature

syntax
tsvFile = emcSampleEntropy(tsvFile, cfg)

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.featureName: str feature name on which the sample entropy will be computed
[OPTIONAL]
*.dispay: boolean deciding if a figure is to be plotted (default: true)

output
tsvFile: MoCap data structure

examples
```
cfg.featureName = 'kinEnerg';
cfg.display = true;
tsvFile = emcSampleEntropy(tsvFile, cfg);
```

comments
feature is saved in tsvFile.processing.entrop

see also
-

## PLOT

### emcPlotCompare

<u>description</u>
Plots the motion capture data of markers selected on the
three axes X, Y, and Z. Can take as many tsvfile as wanted and plot them
in different colors to compare them.

<u>syntax</u>
emcPlotCompare(tsvFile1[, tsvFile2,…], cfg);

<u>input parameters</u>
tsvFile: MoCap data structure
cfg: configuration structure
   [MANDATORY]
   -
   [OPTIONAL]
   *.plotMarkerList: cell array with markers to be plotted (if not defined all markers will be used)

<u>output</u>
-

<u>examples</u>
cfg.plotMarkerList = {'a','b','c'};
emcPlotCompare(tsvFile1, tsvFile2, cfg);

<u>comments</u>
-

<u>see also</u>
-

## PLOT

### emcPlotMarker

description
Plots the motion capture data of different markers selected
in cfg in different colours on the three axis X, Y, and Z

syntax
emcPlotMarker(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
 [MANDATORY]
 -
 [OPTIONAL]
 *.plotMarkerList: cell array with markers to be plotted (if not defined all markers will be used)

output
-

examples
```
cfg.plotMarkerList = {'a','b','c'};
emcPlotMarker(tsvFile, cfg);
```

comments
-

see also
-

## PLOT

### emcPlotQuiver

description
Plots markers, their line of movement in
space as well as arrows for Speed and acceleration using the function
Quiver from Matlab

syntax
emcPlotQuiver(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  -
  [OPTIONAL]
  *.step: int value defining the step between two arrows (default: 20)
  *.plotMarkerList: cell array with markers to be plotted (if not defined all markers will be used)
  *.plotBodyFlag: boolean flag defining if the body structure is
  printed or not
  if cfg.plotBodyFlag: true
   *.connectionMatrix: list of indexes defining the connection of the
   body (see emcPlotBody3D)

output
-

examples
```
cfg.step = 40;
cfg.plotMarkerList = {'a','b','c'};
cfg.plotBodyFlag = true;
cfg.connectionMatrix = [1 2; 2 3; 1 3]
emcPlotQuiver(tsvFile, cfg);
```

comments
-

see also
emcPlotBody3D

## PLOT

### emcPlotBody3D

description
Plots the structure of the body in 3D

syntax
emcPlotBody3D(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.connectionMatrix: list of indexes defining the connection of the body (see emcPlotBody3D)
[OPTIONAL]
*.frame: int value defining the frame at which the marker should be printed (default: random)
*.lineVector: cell array containing pairs of markers defining red lines

output
-

examples
```
cfg.connectionMatrix = [1 2; 2 3; 1 3]
cfg.frame = 40;
cfg.lineVector = {{'a','b'},{'b','c'}};
emcPlotBody3D(tsvFile, cfg);
```

comments
-

see also
-

## PLOT

### emcAnimation

description
Creates an animation of the motion capture data with the function mcanimate from the Motion Capture Toolbox, Copyright 2008, University of Jyvaskyla, Finland

syntax
emcAnimation(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
  [MANDATORY]
  *.connectionMatrix: list of indexes defining the connection of the body (see emcPlotBody3D)
  *.dataMapar: str path pointing at mcdemodata.mat from the Motion Capture Toolbox, Copyright 2008, University of Jyvaskyla, Finland
  [OPTIONAL]
  *.fps: int number of frame per second at which the points will be displayed in the animation (default: 30)
  *.msize: int size at which the points will be displayed (default: 2)
    lines

output
-

examples
```
cfg.connectionMatrix = [1 2; 2 3; 1 3]
cfg.dataMapar = 'C:/test/mcdemodata.mat';
cfg.fps  =  40;
cfg.msize = 3;
emcAnimation(tsvFile, cfg);
```

comments
-

see also
mcanimate

## PLOT

### emcVideoFeature

description
Plots an animation of both the body moving and a defined feature evolving over time

syntax
emcVideoFeature(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.connectionMatrix: list of indexes defining the connection of the body (see emcPlotBody3D)
*.videoFeature: str feature name to be displayed
[OPTIONAL]
*.step: int int number of frames to skip in between two timestamps (default: 50)
*.videoName: str name of the video to be saved
*.videoFrameRate: int framerate at which the video is recorded

output
-

examples
```
cfg.connectionMatrix = [1 2; 2 3; 1 3]
cfg.videoFeature = 'speed';
cfg.step  =  30;
cfg.videoName = 'testVideo.avi';
cfg.videoFrameRate = 45;
emcVideoFeature(tsvFile, cfg);
```

comments
Feature must have been computed beforehand using funciton in Processing Directory

see also
-

# PLOT

## emcVideoFeatureGroup

description
Plots an animation of both the bodies of a group of people moving and a defined feature evolving over time

syntax
emcVideoFeatureGroup(tsvFile1, tsvFile2, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.connectionMatrix: list of indexes defining the connection of the body (see emcPlotBody3D)
*.videoFeature: str feature name to be displayed
[OPTIONAL]
*.step: int int number of frames to skip in between two timestamps (default: 50)
*.videoName: str name of the video to be saved
*.videoFrameRate: int framerate at which the video is recorded

output
-

examples
```
cfg.connectionMatrix = [1 2; 2 3; 1 3]
cfg.videoFeature = 'speed';
cfg.step = 30;
cfg.videoName = 'testVideo.avi';
cfg.videoFrameRate = 45;
emcVideoFeature(tsvFile1, tsvFile2, cfg);
```

comments
Feature must have been computed beforehand using funciton in Processing Directory

see also
-

## PLOT

### emcVisualSequence

description
Plots a sequence of the full movement frame after frame in one image (chronophotographic) and a boxed representation of the magnitude of the defined feature

syntax
emcVisualSequence(tsvFile, cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
*.connectionMatrix: list of indexes defining the connection of the body (see emcPlotBody3D)
*.visSeqFeature: str feature name to be displayed
[OPTIONAL]
*.step: int number of frames to skip in between two timestamps (default: 50)
*.width: str width of a single frame within the chronophotographic (default: 40)

output
-

examples
```
cfg.connectionMatrix = [1 2; 2 3; 1 3]
cfg.visSeqFeature = 'speed';
cfg.step  =  30;
cfg.width = 50;
emcVisualSequence(tsvFile, cfg);
```

comments
-

see also
-

## SAVE

### emcSaveGraph

Description
Saves all the graphs opened and active

syntax
```
emcSaveGraph(outputDir);
```

input parameters
outputDir: str path where to save all graphs

output
-

examples
```
emcSaveGraph('C:/save');
```

comments
Saves every graph using the name given when opening the feature.
Saves graph in pdf format

see also
-

## SAVE

### emcSaveFeature

Description
Save feature into csv files format (one feature per file) &
a summary file

syntax
emcSaveFeature(tsvFile1[, tsvFile2,...], cfg);

input parameters
tsvFile: MoCap data structure
cfg: configuration structure
[MANDATORY]
-
[OPTIONAL]
*.saveSingleFeatureFlag: boolean flag defining is single feature csv
files should be saved
*.saveSummaryFeatureFlag: boolean flag defining is summary csv
file should be saved
*.saveFeatureList = cell array of all the features to be saved (default: all available)
*.summaryType: cell array containing the different type of operation
to be executed while saving the summary file - Features can be either
AVG (averaged), MEDIAN (median), STD (standard deviation), TRIMMEAN
(trim mean), MAX (max value), SUM (summed)
*.saveTrimmeanParam: int value of the cutoff for trimmean (see trimmean)
*.summaryPlane: cell array defining the different plane to compute
the summary on. Creates different summary for each plane. IN
DEVELOPMENT

output
-

examples
```
cfg.saveSingleFeatureFlag = true;
cfg.saveSummaryFeatureFlag = true;
cfg.saveFeatureList = {'speed', 'convexhull'};
cfg.summaryType = {'AVG', 'TRIMMEAN'};
cfg.saveTrimmeanParam = 40;
emcSaveFeature(tsvFile1,tsvFile2, cfg);
```

comments
-

see also
-