A biological motion toolbox for reading, displaying, and manipulating motion capture data in research settings

Jeroen J. A. van Boxtel

Department of Psychology, University of California–Los Angeles, Los Angeles, CA, USA
Faculty of Medicine, Nursing and Health Sciences, Monash University, Clayton Campus, Victoria, Australia

Hongjing Lu

Department of Statistics and Department of Psychology, University of California–Los Angeles, Los Angeles, CA, USA

Biological motion research is an increasingly active field, with a great potential to contribute to a wide range of applications, such as behavioral monitoring/motion detection in surveillance situations, intention inference in social interactions, and diagnostic tools in autism research. In recent years, a large amount of motion capture data has become freely available online, potentially providing rich stimulus sets for biological motion research. However, there currently does not exist an easy-to-use tool to extract, present and manipulate motion capture data in the MATLAB environment, which many researchers use to program their experiments. We have developed the Biomotion Toolbox, which allows researchers to import motion capture data in a variety of formats, to display actions using Psychtoolbox 3, and to manipulate action displays in specific ways (e.g., inversion, three-dimensional rotation, spatial scrambling, phase-scrambling, and limited lifetime). The toolbox was designed to allow researchers with a minimal level of MATLAB programming skills to code experiments using biological motion stimuli.

Introduction

The ability to recognize and interpret movements of the human body is essential for inferring the goals and intentions of other people. The motions created by living organisms are often termed "biological motion," and such motion provides critical cues to convey socially important affective information (Chouchourelou, Matsuka, Harber, & Shiffrar, 2006; Dittrich, Troscianko, Lea, & Morgan, 1996; Roether, Omlor, Christensen, & Giese, 2009) and communicative signals for human interactions (Manera, Schouten, Becchio, Bara, & Verfaillie, 2010; Poizner, Bellugi, & Lutes-Driscoll, 1981). Given its ecological importance in survival and social interactions, humans are exquisitely sensitive in recognizing and interpreting biological motion patterns (e.g., Matsuzaki & Sato, 2008; Pollick, Hill, Calder, & Paterson, 2003), but exactly how the brain achieves this feat is still largely a mystery.

Because of its importance to human functioning, biological motion has been an increasingly important research topic, drawing attention from researchers in fields such as perception, neuroscience, computer vision, and robotics. Just within the field of perception, biological motion research has been conducted in areas as diverse as animal perception, infant studies, and autism research. For example, biological motion has been employed to investigate humans' sensitivity to communicative and social interactions (Chouchourelou et al., 2006; Dittrich et al., 1996; Manera et al., 2010; Poizner et al., 1981; Roether et al., 2009), emotion (Chouchourelou et al., 2006; Dittrich et al., 1996; Hubert et al., 2007; Pollick et al., 2003; Pollick, Paterson, Bruderlin, & Sanford, 2001), sensitivity to different types of action (Dittrich, 1993; van Boxtel & Lu, 2011), developmental research in children (Carter & Pelphrey, 2006; Fox & McDaniel, 1982; Pelphrey & Carter, 2008) and adults (Norman, Payton, Long, & Hawkes, 2004; Pilz, Bennett, & Sekuler, 2010), sensitivity of animals to biological motion (Herman, Morrel-Samuels, & Pack, 1990; Oram & Perrett, 1994; Regolin, Tommasi, & Vallortigara, 2000), autism spectrum disorder (e.g., Blake, Turner, Smoski, Pozdol, & Stone, 2003; Freitag et al., 2008; Herrington et al., 2007; Hubert et al., 2007; Murphy, Brady, Fitzgerald,

Citation: van Boxtel, J. J. A., & Lu, H. (2013). A biological motion toolbox for reading, displaying, and manipulating motion capture data in research settings. *Journal of Vision*, *13*(12):7, 1–16, http://www.journalofvision.org/contents/13/12/7, doi:10.1167/13.12.7.

doi: 10.1167/13.12.7 Received April 12, 2013; published October 15, 2013 ISSN 1534-7362

& Troje, 2009; Parron et al., 2008), and contributions of low- and high-level brain areas to visual perception (Chang & Troje, 2009; Hirai & Kakigi, 2008; Thurman & Lu, 2013; van Boxtel & Lu, 2013). Furthermore, neuroimaging and physiological studies aim to reveal the neurobiological underpinnings of biological motion perception. Significant progress has been made in terms of localizing brain areas specific to biological motion (Grossman et al., 2000; Oram & Perrett, 1994) and social perception (Iacoboni et al., 2004), providing evidence for thought-provoking concepts such as "snapshot" neurons (Vangeneugden, Pollick, & Vogels, 2009) and identifying brain (endo-)phenotypes of e.g., autism spectrum disorder (e.g., Freitag et al., 2008; Herrington et al., 2007; Hubert et al., 2007; Kaiser & Pelphrey, 2012).

Although in recent decades researchers have made impressive advances in understanding the basis for the perception of biological motion, many fundamental questions still remain unanswered. The complexity of stimuli poses one of the difficult issues that researchers confront. There is, as far as we know, no easy way in which a researcher with limited programming skills interested in biological motion perception can take recorded data from motion capture systems and display the action in a research setup. The lack of an easy tool for presenting and manipulating biological motion stimuli motivated us to develop the *BiomotionToolbox*.

Biological motion can be investigated in many ways. One can show movies to observers or static images with postures that imply motion. However, the most popular display in psychophysics, developed by Johansson (1973), is a technique based on point-light actions. Historically, point-light actions were made by placing markers at several important joints of an actor and recording the performed action in darkness, such that only the marked joints were visible. Nowadays, point-light displays (PLDs) are generally created on a computer, using video or motion capture input. The main advantage of PLDs is that they contain mainly motion information and very little extraneous form information about human body structure or other cues such as facial expressions, shadows, hair, or clothing. One also has considerable freedom in manipulating the marker motions, for example by inverting them (Pavlova & Sokolov, 2000; Sumi, 1984) or by changing the spatial layout (Bertenthal, Proffitt, & Kramer, 1987). The BiomotionToolbox is specifically designed to display and manipulate PLDs, allowing for specific manipulations with a single call to a function.

A popular way to create PLD data is to capture an action with a video camera and convert this data to PLD manually (or sometimes automatically) by extracting the joints from each movie frame and recording the positions so they can be replayed later in an experiment. As well as being labor intensive, this method restricts the potential uses of PLDs because the joint location can only be captured two-dimensionally and thus it cannot be viewed from other viewing angles. In addition, joints may be invisible due to occluding body parts, and the PLD can only be displayed from one viewing distance (the original recording distance).

In recent years, it has become possible to overcome these problems by employing motion capture data (MCD). MCD can be acquired in several different ways, but the main idea is to use three or more cameras (rather than just one) to record the motions of marked joints in three-dimensional (3-D) space, plus time. With this setup, each joint can be followed continuously, and in 3-D, thus circumventing the problems associated with a single-camera setup. The disadvantage of this method is that it requires specialized equipment and a dedicated recording space. Therefore, most researchers do not have a motion capture device at their disposal. However, because of its increasing popularity, more MCD has become available online, with several of these databases being free (see Appendix III for a selection). These databases contain a wide range of actions and should give most researchers enough flexibility to select appropriate biological motion stimuli in order to address the questions in which they are interested.

Unfortunately, for many researchers the use of these online motion capture databases is difficult because they lack the knowledge of how to convert the files from motion capture system into useable formats that can be displayed on a computer screen for psychophysical experiments. In addition, rotating, inverting or scrambling the MCD may pose problems. Many researchers may be deterred by these daunting hurdles, causing many potentially fruitful research ideas not to be pursued.

The BiomotionToolbox we developed takes most of these tasks out of the hands of the researchers and automates them; the BiomotionToolbox allows researchers to read a variety of motion capture data, then write and manipulate biological motion data obtained from motion capture devices. In combination with the Psychtoolbox 3.0 (Brainard, 1997; Pelli, 1997, the BiomotionToolbox allows for easy-to-program biological motion research in the MATLAB® environment (The MathWorks, Inc., Natick, MA). The BiomotionToolbox is designed to allow people with minimal programming skills to create experiments with complex biological motion stimuli. We have employed an early version of the toolbox in a recent study (van Boxtel & Lu, 2013). Partial forms of the toolbox were used by van Boxtel and Lu (2011, 2012).

In the following sections we will discuss the central functions of the BiomotionToolbox. We explain how to set up the toolbox, how to initialize biological motion objects, how to manipulate them (e.g., inverting and

rotating), and how to obtain the data required to display them.

Definitions

The BiomotionToolbox is written around a MATLAB handle class. An instance of such a class (as created during the Initialization, explained below) is called an object. With the purpose of the toolbox in mind, one can think of an object as a biological motion actor. In the subsequent sections we will therefore often refer to such an object as a biological motion object, often called `myMO` (short for my motion object) in our example coding programs.

Each object has several properties (i.e., settings or parameters) that describe the state of the object (e.g., if it is inverted). Similarly, objects have associated functions (called methods) that operate on the data contained in the object. Using these methods, one can access and alter the biological motion data.

Parameters and methods can be read or invoked with the dot-operator. For example, when one wants to know if an actor is inverted, one can call `myMO.Invert`, and when one wants to invert the actor, one can call `myMO.Invert = 1`. Methods work the same way. For example, one can call the function `SmoothLoop` as follows: `myMO.SmoothLoop`. See the additional codes (Appendix II) for examples.

Getting started

Getting started is easy. Download the toolbox from http://www.jeroenvanboxtel.com/software/BioMotionToolbox.php, and add the BiomotionToolbox folder to the path in MATLAB.

Supported file formats

The BiomotionToolbox was designed to read motion capture data from various sources. Motion capture can be saved in various formats (e.g., bvh, c3d, and ASF/AMC). Some research groups have made motion capture data available in custom formats (Ma, Paterson, & Pollick, 2006; Vanrie & Verfaillie, 2004). The BiomotionToolbox currently supports several of these formats (see Table 1 for supported formats) and uses one of them as a default (data3d). The c3d format is read with an example code provided on the motion capture site of Carnegie Mellon University (http://mocap.cs.cmu.edu/tools.php).

The data3d format

The BiomotionToolbox accepts several file types, but its native format is data3d. This is a text file that contains the motion capture data in the following format:

```
Time1_Joint1_x Time1_Joint2_x Time1_Joint3_x ...
Time1_Joint1_y Time1_Joint2_y Time1_Joint3_y ...
Time1_Joint1_z Time1_Joint2_z Time1_Joint3_z ...
Time2_Joint1_x Time2_Joint2_x Time2_Joint3_x ...
Time2_Joint1_y Time2_Joint2_y Time2_Joint3_y ...
Time2_Joint1_z Time2_Joint2_z Time2_Joint3_z ...
.....
```

Each column contains the position information for one joint, and each triplet of rows defines the *x*-, *y*-, and *z*-coordinates in 3-D.

Using 2-D data

In case a user wants to use the BiomotionToolbox on previously acquired 2-D data, those data can be imported, with all *z*-coordinates set to zero, and saved as a txt file in the data3d format. Then one can initialize a biological motion object as if it were 3-D and make use of all the functions of the BiomotionToolbox. Missing data (e.g., due to occlusion) should be denoted with `NaN` (not a number) for *x*-, *y*-, and *z*-coordinates.

Initialization

Basic initialization

A user needs to initialize a Biomotion object (i.e., an action) before using it. In its most basic instantiation, initialization requires only a filename that contains 3-D motions of all joints.

```
myMO = BioMotion('filename');
```

This command will open the file `filename` and automatically initializes all the necessary parameters in the Biomotion object `myMO`.

The BiomotionToolbox can often recognize what the input format of motion capture data is (e.g., bvh or c3d), but this function strongly depends on the use of the correct file extension (see Table 1 for supported formats). If your file has a nonstandard file extension but the file nonetheless contains readable motion information, you need to provide the type of input in the optional properties (see "Options" below).

Extension	File type	Description
ptd	ptd or pollick	Files from the Pollick lab Body Movement Library
txt	vanrie	Text files from the Leuven Action Database
c3d	c3d	c3d files
bvh	bvh	bvh files
txt	data3d	Text file that is in the format natively supported by the BiomotionToolbox. It is formatted as the transpose of JointsInfo.

Table 1. Supported file formats. Both 'vanrie' and 'data3d' file types have the extension txt. The BiomotionToolbox will recognize the difference between them based on the content of the files, or by the user-provided file type.

Options

The initialization routine has several options that will allow the user to customize the initialization of the input data. The options allow the user to define the file type, to anchor the Biomotion object to a certain location, and to read a subset of the joints (markers) or frames from the motion capture data.

The optional customizations should be entered as follows:

```
myMO = BioMotion('filename', 'PropertyName1', 'PropertyValue1', ...);
```

Behind filename, one or more pairs of 'PropertyNames' (in quotes), and PropertyValues should follow. PropertyName is the name of one of the optional properties that the user likes to set, and PropertyValue is the value that the user would like to assign to that property. Possible property names are 'Filetype', 'Anchor', 'SelectJoints', and 'SelectFrames', which we will discuss below.

Filetype

To prevent potential problems reading the files, or to clearly define the file type when using a nonstandard extension, a user can employ the property Filetype. The user can manually provide the file type of the imported data in PropertyValue. Options are 'ptd', 'pollick', 'vanrie', 'c3d', 'bvh', and 'data3d' (see Table 1). When Filetype is not provided, the program attempts to determine the file type automatically, mostly based on the file extension.

Anchor

The option Anchor allows the user to select a set of joints, which will function as a spatial "anchor" for the biological motion. Anchoring here means that the average position of all joint numbers provided as PropertyValues will be repositioned to [0, 0, 0]. This allows one, for example, to display a Biomotion walker to walk "on a treadmill" in the center of the screen. Note that the PropertyValues for Anchor are relative to the input file and not relative to the joints selected with SelectJoints.

SelectJoints

This option provides a way to select a subset of the joints from the input file by providing their indices in an array in the PropertyValue. For example, many of the bvh actions in the Carnegie Mellon Database (created with the amc2bvh program) provide several dozens of joint markers, and in order to create a standard actor with 13 joints, a user needs to select those joints. For most bvh files in this particular database, the 13 joints often are associated with the following indices, [17 19 20 21 26 27 28 3 4 5 8 9 10], respectively corresponding to Head, Shoulder1, Elbow1, Wrist1, Shoulder2, Elbow2, Wrist2, Hip1, Knee1, Ankle1, Wrist2, Knee2, Ankle2. In this case, a user can define the option as

```
myMO = BioMotion('filename', 'SelectJoints', [17 19 20 21 26 27 28 3 4 5 8 9 10]);
```

SelectFrames

This option provides a way to select a subset of the frames from the input file by providing their indices in an array in the PropertyValue. Note that the frames are played in the order in which they appear in PropertyValue. Thus, BioMotion('filename', 'SelectFrames', 1:10) will take the first ten frames from the input file. BioMotion ('filename', 'SelectFrames', [1 50 29 67 26]) will take the requested frames *in the provided order*. If you want to play them in ascending/descending order you will need to sort the array before you provide them as input (e.g., using the MATLAB function sort).

Empty initialization

A user can also initialize an empty Biomotion object, by calling BioMotion without arguments: myMO = BioMotion();

Initialization from an array

When a user already has motion capture data in an array and wants to create a Biomotion object, the user can still use the same initialization step as above. To do so, instead of providing a file name, the user can directly input an array. This array must follow the format of `NormJointsInfo`, i.e., [number of joints × number of frames × 3 (*x*-, *y*-, *z*-coordinates)], otherwise an error message will be returned (or the action will not display properly). For example, `BioMotion(array, 'SelectFrames', [1 50 29 67 2 6])`.

Making arrays of Biomotion objects

It is often useful to generate an array of different actions. This will allow the use of `SetAll` and `SetAllVect` (see Toolbox methods), for fast and easy setting of properties. The main function for reading data, `GetFrame`, also works with an array of Biomotion objects.

An array of Biomotion objects can be created as follows:

```
bmArray(1) = BioMotion('filename1');
bmArray(2) = BioMotion('filename2');
```

or

```
bm1 = BioMotion('FileWithBm1Data');
bm2 = BioMotion('FileWithBm2Data');
bmArray = [bm1 bm2];
```

Both these codes create an array of two Biomotion objects. Note that initializing one Biomotion object (`bm1`) and then typing `bmArray = [bm1 bm1]` will also create a Biomotion array, but in this case the two items are "yoked," that is, if a property is changed in one object, it will be changed in the other too. This characteristic is a property of the handle class in MATLAB. If a user wants to create an array of independent copies of bm1, the user will need to use the either of the two above-explained initializations or use the `Copy` method (see below).

Copying Biomotion objects

A user should not copy Biomotion objects using the "is equal" sign (=). For example, do not do something like `bm2 = bm1`. Because the Biomotion object is a "handle" class in MATLAB, `bm2` will not just be a simple copy. Instead, the command with the equal sign provides a "link" (handle) to `bm1`, and therefore anything that is changed in `bm2` will *also* change in `bm1`, and vice versa (i.e., they are yoked).

To allow copying of the data from one object to another, the BiomotionToolbox includes a method that ensures independence of both objects. This is the `Copy` function, which copies all the properties (both public and private) of one object to another object. To copy object A into object B, call `B = A.Copy;`

In order to make an array of copies of `bm1` (as attempted above) one can type the following: `bmArray = [bm1 bm1.Copy];`.

Note that `Copy` is relatively slow (on the order of 100 ms per copy). Therefore, do not copy objects in time-constrained situations (as when attempting to update every monitor frame).

Toolbox properties

After initialization, an object generated with the BiomotionToolbox (e.g., `myMO`) will contain several fields (properties). Most of these fields are set after initialization, but can be modified by the user. Other properties are read-only.

Changing these properties will set often-used manipulations of the biological motion stimuli, including inversion, rotation, spatial scrambling, phase scrambling, and limited lifetime manipulations. For example, inverting an action is as simple as calling

```
myMO.Invert = 1
```

These properties, and the user's control over them are the core value of the BiomotionToolbox. An overview of the toolbox properties and their description is provided in Table 2 and Table 3. A more detailed description of how to use these properties is provided in Appendix I.

Toolbox methods

The objective of the BiomotionToolbox is to allow easy manipulation and displaying of motion-capture data in a point-light display. A central method of the BiomotionToolbox is `GetFrame`, which allows a user to access (read) the motion capture data. To access the 3-D positions of all joints in a single frame, a user can call `myMO.GetFrame(n)`, which will output the joint positions of `myMO` at frame n (returning a 3 × `nPointLights` array). If `myMO` is an array of objects, `GetFrame` will a return the joints at frame n of all objects. These data can then be displayed with the function `moglDrawDots3D` from the PsychToolbox (see examples in Appendix II).

By default, `GetFrame` will return the 3-D coordinates of all the joints for the requested frame number, but a user can also specify a subset of joints. Both the

Read only fields	
`nPointLights`	Number of point lights (= markers) making up the actor.
`nFrames`	Number of frames in the action sequence.
`Filename`	Contains the name of the file that is used to generate the Biomotion object.
`Filetype`	Contains the extension (or the file type) of the file used to generate the Biomotion object.
`JointsInfo`	`JointsInfo` contains the 3-D joint information in the following format: column x_positions time 1, column y_positions time 1, column z_positions time 1, column x_positions time 2, ... etc. The size of this 2-D array is [nPointLights × (3 × nFrames)].
`NormJointsInfo`	`NormJointsInfo` contains the 3-D joint information in the format [npointlights × nframes × 3 [x y z dimensions]] after normalization.
`AnchorJoints`	`AnchorJoints` contains the joints that were used to calculate the "center of mass" of the actor. The average position of the `AnchorJoints` joints will be at [0, 0, 0] in 3-D space, e.g., a user can ensure a "treadmill" walk by providing the indices of the hip joints. For several input files the default values are the two hip joints (i.e., [8 11] for any data3d file), [10 13] for ptd files, and [8 9] for vanrie files. For bvh and c3d files, the default is to use all joints. A user also can input `'none'`, which will prevent the joint array from being anchored. If a user does not want to use the default values, the user needs to provide these numbers as property values of `'Anchor'` when a Biomotion object is initialized. Note, however, that the indices in `AnchorJoints` are relative to the current Biomotion object, while the property values in 'Anchor' should be defined relative to the input file.
`InfoLimitedLife`	`InfoLimitedLife` is an object from the limitedlifetime class, and contains information about the limited lifetime properties. `InfoLimitedLife` is empty by default, but when LimitedLife is set to 1, it is created. It then contains the following fields: `maxlife`: the maximum lifetime of each marker `ndots`: the number of markers that will be drawn (must be less than or equal to the number of skeleton segments in myOM.Skeleton `type`: the type of refresh. Can be `'async'` and `'sync'`. If `'async'` all the dots are initialized with a random age (≤ `maxlife`), and will thus be refreshed asynchronously. If `'sync'`, all dots will be initialized with age 1, and thus will be refreshed synchronously. `takedots`: logic array that indicates which markers are selected to be displayed (1) or not (0).

Table 2. Read-only properties.

frame number and the joints can be separately specified for different objects in an object array (see Appendix I). For many applications `GetFrame` is the only function (apart from initialization) that one will use (see Appendix II for example programs).

To increase the efficiency of manipulating parameters in Biomotion stimuli, two methods are included in the toolbox to allow quick setting of parameters in an array of Biomotion objects. These functions are `SetAll` and `SetAllVect`, which allow one to set one property value (e.g., `Scramble`), or an array (e.g., `Position3D`) for several biological motion objects at the same time (see Appendix I).

A convenient way to rotate an entire Biomotion object is to use `RotatePath`. `RotatePath(val)`, rotates the entire movie by the amount specified in `val` (in radians). The rotation axis is defined in `myMO.RotationAxis`. This function achieves the same as calling `myMO.Rotation = val`, but does it on all frames simultaneously, and changes `NormJointsInfo`. `myMO.Rotation` performs the rotation on each frame as you call `GetFrame`. Thus `RotatePath` may be faster in certain situations. For example it is useful in cases where it turns out that your input file contains the joint information upside-down. To put the action upright, set `myMO.RotationAxis = [1 0 0]` (rotation around *x*-axis), and then `myMO.RotatePath(pi)`.

Sometimes the looping of the movie is not perfect (the transition from the last frame back to the first is not smooth). `Smoothloop` may help in smoothing the transition when looping an action. `Smoothloop` will calculate the linear distance between the position of each joint in the first and last frame, and then add an increasingly bigger offset to successive frames to

Read and write fields	
`Invert`	Invert (1) the action, or not (0; default)
`Scramble`	Space scramble (1) the action, or not (0; default)
`Rotation`	Rotation angle (radians; 0 is default). The rotation is relative to the current orientation.
`RotationAxis`	Rotation axis. The axis [*x y z*] around which the rotation is performed. e.g., in order to rotate around the *x*-axis, put RotationAxis to [1 0 0]. Default is [0 1 0], rotation in depth.
`ScrambleDim`	When (location) scrambling the action, only the dimensions indicated here will be scrambled. Default '*xyz*' can be changed to '*y*' or '*xz*', etc.
`ScrambleWidth`	Maximum scramble amount in *x* direction (default is max span in *x* direction over the duration of the movie)
`ScrambleHeight`	Maximum scramble amount in *y* direction (default is max span in *y* direction over the duration of the movie)
`ScrambleDepth`	Maximum scramble amount in *z* direction (default is max span in *z* direction over the duration of the movie)
`ScrambleOffsets`	Array of size [3 × nPointLights] with [*x y z*] scramble amount for each joint. Will be set automatically when setting Scramble to 1. But can also be set manually (*before* setting Scramble to 1), either by inputting it directly, or by calling `ScramblePositions` after setting `ScrambleWidth`, etc., to the desired values.
`Position3D`	Displace the 3-D location of the actor by a certain amount. Input a 1 × 3 vector with *x*, *y*, and *z* values ([*x y z*]).
`Scale`	Scale the actor. This scaling is relative to the current size, so if you call it twice in a row, the actor will increase twice. e.g., calling `BM.Scale = 2;` and then again BM.Scale = 2; will result in a four-times larger actor.
`Loop`	Loop the movie when reaching the end of the action sequence. 1 = yes or 0 = no (default).
`LimitedLife`	Display the action as a limited lifetime stimulus (1) or not (0; default).
`PhaseScramble`	Phase scramble the Biomotion object (0 [default] or 1).
`PhaseOffsets`	A [1 × nPointLights] array with the phase offsets (in number of frames) for each joint of the Biomotion object. This will be set automatically when you set PhaseScramble to 1, but there is an option to set it manually.
`Skeleton`	Defining e.g., an arm that goes from marker 2 to 3 and then to 4, you will define the skeleton as [2 3; 3 4; ...]. The ellipses (...) is where the other limb segments will be defined. It is essential to set this property when using limited lifetime markers that need to be drawn on the skeleton (and not just on the joints).
	When setting `LimitedLife` to 1, `Skeleton` will be set to [1 1; 2 2; 3 3; ...], that is the markers will be drawn on the joints.

Table 3. Readable and writable properties.

completely cancel out the spatial differences by the last frame. This function will only work on differences between the first and last frame and will not remove a deviant value in an intermediate frame.

`SaveData3D` allows you to save the current PLD in data3d format. This is useful, for example, when you initialize with a bvh or c3d file, which are slow to import. Saving the data in data3d format, and then using that file to initialize the biological motion object in the experimental program will be much faster. `SetLimitedLifetimeParams` can be used to change the parameters of limited lifetime stimuli. Users can change the number of dots (`ndots`) and the dot lifetime (`maxlife`). Before, using this function, set the `LimitedLife` to 1.

Troubleshooting

We have tried to provide sufficient error feedback to allow the user to identify the error in the code in most cases. The most common error is that an out-of-bounds (e.g., too large) frame number is requested. If this is the case, you will receive an error message that identifies this error. You can solve this problem by setting `Loop` to 1. We have not opted to make 1 as the default value

of `Loop`, because this would never produce an error, making a programming mistake hard to spot when Looping is not desired.

Performance increase

The BiomotionToolbox is structured around a handle class in MATLAB. Handle classes are somewhat slower than, e.g., a struct or an old-style class definition in MATLAB. However, we have found the speed to be sufficiently fast to compute and display at least three PLDs simultaneously in real time on the slowest computer in our lab. Other setups (e.g., an Intel core 2 quad q6600 @ 2.40 GHz, with 3.5 GB RAM, and ATI Radeon X1600 Series graphics card) could display at least 15 objects simultaneously and in real time. Of course speed depends on various variables, including what other programs are running on the computer, your graphics card, and what else you display on the screen.

There are ways to increase performance during real-time computation in BiomotionToolbox. Change properties as little as possible (e.g., set Invert before the experiment, and not on every frame). Use RotatePath when you want to apply the rotation to every frame, rather than using the Rotation property (which implements a rotation on every call to GetFrame and is therefore slower).

Initializing a Biomotion object from a c3d or bvh file takes a long time. It may be better to create data3d files from the c3d or bvh files offline and use those files from then on. This can be done with the method `SaveData3D`.

Another way to speed up a program is to use an alternative way to call methods. In the examples we use the dot operator, but it is also possible to pass a Biomotion object as an argument to a function, which is faster (especially on older versions of MATLAB). Thus, instead of calling `myMO.GetFrame(1)` to get frame 1, a user can call `GetFrame(myMO,1)` to speed up the program.

Limitations

Currently a limitation is the number of input formats that the BiomotionToolbox currently accepts. Motion capture data is available in many different formats, and we have chosen to implement some of the most prevalent, including two (`vanrie` and `ptd`) that have been used in recent papers (Ma et al., 2006; Vanrie & Verfaillie, 2004). Still, some formats are missing (notably the ASF/AMC format), and we hope that future versions of the BiomotionToolbox will accept more formats. Readers can check for updates at http://www.jeroenvanboxtel.com/software/BioMotionToolbox.php or http://cvl.psych.ucla.edu/resources.htm.

Final remarks

We have included some sample code to show a point-light stimulus (example 1), a scrambled actor (example 2), a limited lifetime example (example 3; Appendix II) and a set of motion-capture databases to get a user started (Appendix III). The sample code will help users learn how to manipulate and display data using the BiomotionToolbox. We hope that this toolbox and the other information provided in this paper (e.g., the motion capture databases) will facilitate research on biological motion.

Keywords: biological motion, motion capture, toolbox

Acknowledgments

This project was supported by a grant from the National Science Foundation (NSF BCS-0843880). The motion-capture data used in the sample code was obtained from http://mocap.cs.cmu.edu, which was created with funding from NSF EIA-0196217.

Commercial relationships: none.
Corresponding author: Jeroen J. A. van Boxtel.
Email: j.j.a.vanboxtel@gmail.com.
Address: School of Psychology and Psychiatry, Monash University, Victoria, Australia.

References

Beintema, J. A., & Lappe, M. (2002). Perception of biological motion without local image motion. *Proceedings of the National Academy of Sciences, USA*, *99*(8), 5661–5663.

Bertenthal, B. I., Proffitt, D. R., & Kramer, S. J. (1987). Perception of biomechanical motions by infants: Implementation of various processing constraints. *Journal of Experimental Psychology: Human Perception & Performance*, *13*(4), 577–585.

Blake, R., Turner, L. M., Smoski, M. J., Pozdol, S. L., & Stone, W. L. (2003). Visual recognition of biological motion is impaired in children with autism. *Psychological Science*, *14*(2), 151–157.

Brainard, D. H. (1997). The Psychophysics Toolbox. *Spatial Vision*, *10*(4), 433–436.

Carter, E. J., & Pelphrey, K. A. (2006). School-aged children exhibit domain-specific responses to biological motion. *Social Neuroscience*, *1*(3–4), 396–411.

Chang, D. H., & Troje, N. F. (2009). Characterizing global and local mechanisms in biological motion perception. *Journal of Vision*, *9*(5):8, 1–10, http://www.journalofvision.org/content/9/5/8, doi:10.1167/9.5.8. [PubMed] [Article]

Chouchourelou, A., Matsuka, T., Harber, K., & Shiffrar, M. (2006). The visual analysis of emotional actions. *Social Neuroscience*, *1*(1), 63–74.

Dittrich, W. H. (1993). Action categories and the perception of biological motion. *Perception*, *22*(1), 15–22.

Dittrich, W. H., Troscianko, T., Lea, S. E., & Morgan, D. (1996). Perception of emotion from dynamic point-light displays represented in dance. *Perception*, *25*(6), 727–738.

Fox, R., & McDaniel, C. (1982). The perception of biological motion by human infants. *Science*, *218*(4571), 486–487.

Freitag, C. M., Konrad, C., Haberlen, M., Kleser, C., von Gontard, A., Reith, W., et al. (2008). Perception of biological motion in autism spectrum disorders. *Neuropsychologia*, *46*(5), 1480–1494.

Guerra-Filho, G., & Biswas, A. (2012). The human motion database: A cognitive and parametric sampling of human motion. *Image & Vision Computing*, *30*(*3*), 251–261.

Grossman, E., Donnelly, M., Price, R., Pickens, D., Morgan, V., Neighbor, G., et al. (2000). Brain areas involved in perception of biological motion. *Journal of Cognitive Neuroscience*, *12*(5), 711–720.

Herman, L. M., Morrel-Samuels, P., & Pack, A. A. (1990). Bottlenosed dolphin and human recognition of veridical and degraded video displays of an artificial gestural language. *Journal of Experimental Psychology, General*, *119*(2), 215–230.

Herrington, J. D., Baron-Cohen, S., Wheelwright, S. J., Singh, K. D., Bullmore, E. T., Brammer, M., et al. (2007). The role of MT+/V5 during biological motion perception in Asperger syndrome: An fMRI study. *Research in Autism Spectrum Disorders*, *1*(1), 14–27.

Hirai, M., & Kakigi, R. (2008). Differential cortical processing of local and global motion information in biological motion: An event-related potential study. *Journal of Vision*, *8*(16):2, 1–17, http://www.journalofvision.org/content/8/16/2, doi:10.1167/8.16.2. [PubMed] [Article]

Hubert, B., Wicker, B., Moore, D. G., Monfardini, E., Duverger, H., Da Fonseca, D., et al. (2007). Brief report: Recognition of emotional and non-emotional biological motion in individuals with autistic spectrum disorders. *Journal of Autism & Developmental Disorders*, *37*(7), 1386–1392.

Iacoboni, M., Lieberman, M. D., Knowlton, B. J., Molnar-Szakacs, I., Moritz, M., Throop, C. J., et al. (2004). Watching social interactions produces dorsomedial prefrontal and medial parietal BOLD fMRI signal increases compared to a resting baseline. *Neuroimage*, *21*(3), 1167–1173.

Johansson, G. (1973). Visual perception of biological motion and a model for its analysis. *Perception & Psychophysics*, *14*(2), 201–211.

Kaiser, M. D., & Pelphrey, K. A. (2012). Disrupted action perception in autism: Behavioral evidence, neuroendophenotypes, and diagnostic utility. *Developmental Cognitive Neuroscience*, *2*(1), 25–35.

Ma, Y., Paterson, H. M., & Pollick, F. E. (2006). A motion capture library for the study of identity, gender, and emotion perception from biological motion. *Behavior Research Methods*, *38*(1), 134–141.

Manera, V., Schouten, B., Becchio, C., Bara, B. G., & Verfaillie, K. (2010). Inferring intentions from biological motion: A stimulus set of point-light communicative interactions. *Behavior Research Methods*, *42*(1), 168–178.

Matsuzaki, N., & Sato, T. (2008). The perception of facial expressions from two-frame apparent motion. *Perception*, *37*(10), 1560–1568.

Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., & Weber, A. (2007). Documentation Mocap Database HDM05. (Tech. Rep. No. CG-2007-2, ISSN 1610-8892). Universität Bonn, Bonn, Germany.

Murphy, P., Brady, N., Fitzgerald, M., & Troje, N. F. (2009). No evidence for impaired perception of biological motion in adults with autistic spectrum disorders. *Neuropsychologia*, *47*(14), 3225–3235.

Norman, J. F., Payton, S. M., Long, J. R., & Hawkes, L. M. (2004). Aging and the perception of biological motion. *Psychology and Aging*, *19*(1), 219–225.

Oram, M., & Perrett, D. (1994). Responses of anterior-superior temporal polysensory (STPa) neurons to "biological motion" stimuli. *Journal of Cognitive Neuroscience*, *6*, 99–116.

Parron, C., Da Fonseca, D., Santos, A., Moore, D. G., Monfardini, E., & Deruelle, C. (2008). Recognition of biological motion in children with autistic

spectrum disorders. *Autism: The International Journal of Research & Practice*, *12*(3), 261–274.

Pavlova, M., & Sokolov, A. (2000). Orientation specificity in biological motion perception. *Perception and Psychophysics*, *62*(5), 889–899.

Pelli, D. G. (1997). The VideoToolbox software for visual psychophysics: Transforming numbers into movies. *Spatial Vision*, *10*, 437–442.

Pelphrey, K. A., & Carter, E. J. (2008). Brain mechanisms for social perception: Lessons from autism and typical development. *Annals of the New York Academy of Sciences*, *1145*, 283–299.

Pilz, K. S., Bennett, P. J., & Sekuler, A. B. (2010). Effects of aging on biological motion discrimination. *Vision Research*, *50*(2), 211–219.

Poizner, H., Bellugi, U., & Lutes-Driscoll, V. (1981). Perception of American sign language in dynamic point-light displays. *Journal of Experimental Psychology: Human Perception & Performance*, *7*(2), 430–440.

Pollick, F. E., Hill, H., Calder, A., & Paterson, H. (2003). Recognising facial expression from spatially and temporally modified movements. *Perception*, *32*(7), 813–826.

Pollick, F. E., Paterson, H. M., Bruderlin, A., & Sanford, A. J. (2001). Perceiving affect from arm movement. *Cognition*, *82*(2), B51–B61.

Regolin, L., Tommasi, L., & Vallortigara, G. (2000). Visual perception of biological motion in newly hatched chicks as revealed by an imprinting procedure. *Animal Cognition*, *3*, 53–60.

Roether, C. L., Omlor, L., Christensen, A., & Giese, M. A. (2009). Critical features for the perception of emotion from gait. *Journal of Vision*, *9*(6):15, 1–32, http://www.journalofvision.org/content/9/6/15, doi:10.1167/9.6.15. [PubMed] [Article]

Sumi, S. (1984). Upside-down presentation of the Johansson moving light-spot pattern. *Perception*, *13*(3), 283–286.

Thurman, S. M., & Lu, H. (2013). Complex interactions between spatial, orientation, and motion cues for biological motion perception across visual space. *Journal of Vision*, *13*(2):8, 1–18, http://www.journalofvision.org/content/13/2/8, doi:10.1167/13.2.8. [PubMed] [Article]

van Boxtel, J. J., & Lu, H. (2011). Visual search by action category. *Journal of Vision*, *11*(7):19, 1–14, http://www.journalofvision.org/content/11/7/19, doi:10.1167/11.7.19. [PubMed] [Article]

van Boxtel, J. J., & Lu, H. (2012). Signature movements lead to efficient search for threatening actions. *PLoS ONE*, *7*(5), e37085.

van Boxtel, J. J. A., & Lu, H. (2013). Impaired global, and compensatory local, biological motion processing in people with high levels of autistic traits. *Frontiers in Psychology, 4, 209*.

Vangeneugden, J., Pollick, F., & Vogels, R. (2009). Functional differentiation of macaque visual temporal cortical neurons using a parametric action space. *Cerebral Cortex*, *19*(3), 593–611.

Vanrie, J., & Verfaillie, K. (2004). Perception of biological motion: A stimulus set of human point-light actions. *Behavior Research Methods, Instruments, & Computers*, *36*(4), 625–629.

Appendix I: Additional explanation properties and methods

Here we assume that you initialized a Biomotion object that you named `myMO`, or an array of Biomotion objects, that you named `myMOs`.

Properties

- `myMO.Invert = val`
  Invert the action. 0 = no (default), 1 = yes, provided in `val`.
- *`myMO.Scramble = val`*
  Space scramble the action. 0 = no (default), 1 = yes. The start locations of the individual joints will be randomly relocated within a space defined by `ScrambleWidth`, `ScrambleHeight`, and `ScrambleDepth`, and saved in `ScrambleOffsets` (a 3 × nPointLights array). If not defined by you `ScrambleOffsets` will be automatically set (after you set Scramble = 1). The size of the space will be bounded by the maximum positions in *x*, *y*, and *z* over the entire movie. You can set the size of the scramble space by entering values for `ScrambleWidth`, `ScrambleHeight`, and `ScrambleDepth` [see Custom scambling in Appendix II].
- *`myMO.ScrambleDim = 'xyz'`*
  Determines which dimension should be scrambled. e.g., `myMO.ScrambleDim = 'xz'`, scrambles only the *x* and *y* dimensions.
- *`myMO.Rotation = val`*
  Rotates the action by the amount specified in val (in radians), relative to the current rotation.
- *`myMO`*`.Position3D=[x,y,z]`
  The position of the action will be offset by the amounts specified in [x,y,z].
- *`myMO.Scale = val`*
  Scale the size of the action. The default value is set to work relatively well for many applications, but

you probably want to resize your actions with this command. This scaling is relative to the current size, so if you call it twice in a row, the actor will increase twice. e.g., Calling `BM.Scale = 2;` and then again `BM.Scale = 2;` will result in a four times bigger actor.

- *myMO.Loop = val*
  Set `val` to 1 if you want to Loop the action once you reach the last frame (default is 0). Setting `Loop` to 1 will cause `GetFrame` to return to the first frame after you have reached the last frame.
- *myMO.LimitedLife = val*
  Set `val` to 1 if you want a limited lifetime action (default is 0). After setting `LimitedLife` to 1, the default settings will produce an action with a lifetime of 1 frame. The markers will be drawn along the skeleton provided in `myMO.Skeleton`. By just providing the skeleton, and setting `LimitedLife` to 1, one obtains the central condition used by Beintema and Lappe (2002). To change the settings for limited lifetime stimuli, use `SetLimitedLife Parameters`.
- *myMO.PhaseScramble = val*
  `val` can be 0, 1, 2 or 'rescramble', or 3. 1 will scramble the phase of each joint individually, meaning that each joint will start at a random frame within the movie. 0 will remove the scrambling. 2 and `'rescramble'` will rescramble. When you call `PhaseScramble = 1` multiple times in a row, each joint will be offset again by the values already set in `PhaseOffsets` (without redrawing them). When you call `PhaseScramble = 2, PhaseOffsets` will be redrawn first, and then these values will be applied to the currently present offsets. Setting 3 will force the use of user-defined `PhaseOffsets`, when they are set after setting `PhaseScramble = 1`. Note: `PhaseScramble` will change the values in `NormJointsInfo` (unlike Scramble, Invert, and Rotation). Reset everything by calling `myMO.PhaseScramble = 0`.
- *myMO.Skeleton = val*
  This property is used to describe the skeleton along which markers should be drawn for limited lifetime stimuli. Each limb segment should be entered separately. If one wants to draw two arms from the biomotion object, and one arm runs from joint 2, to 3, to 4, and the other arm runs from 5, to 6, to 7, then one should enter: `myMO.Skeleton = [2 3; 3 4; 5 6; 6 7];`

Methods

- `myMO.`**`GetFrame`**`(frame_nmbr [,joints])`
  `GetFrame` will return a `[3 × nPointLights]` array with the [*x, y, z*] positions of the joints at the frame provide by an integer: `frame_nmbr`. If you only want the information from specific joints, you can provide an array with those joint indices in `joints`.
  If `myMO` is an array of Biomotion objects, and `frame_nmbr` is a single integer, `GetFrame` will return the joint information in frame `frame_nmbr` for all objects in myMO. If `frame_nmbr` is an array of the same length as `myMO`, then `GetFrame` will return the joint information in the frame numbers provided for each object. So if `myMO` is of length 2, and you call `myMO.GetFrame([1 30])`, the joint information of the first object in `myMO` will be returned for frame 1, and for frame 30 for the second object in `myMO`. Similarly, you can ask for different joints for the different objects: e.g., `myMO.GetFrame([1 30], [1 2 3; 6 7 8])`, will return joint 1, 2, and 3 on frame 1 for the first object in myMO, and joint 6, 7, and 8 on frame 30 for the second object.
- `myMO.`**`RotatePath`**`(val)`
  `RotatePath` rotates the entire movie by the amount specified in `val` (in radians). The rotation axis is defined in `myMO.RotationAxis`. This function achieves the same as calling `myMO.Rotation=val`, but does it on all frames simultaneously and changes `NormJointsInfo`. `myMO.Rotation` performs the rotation on each frame as you call `GetFrame`. Thus `RotatePath` may be faster in certain situations; it is also useful if it turns out that your input file contains the joint information upside-down. To put the action upright, set `myMO.RotationAxis=[1 0 0]` (rotation around *x*-axis), and then `myMO.RotatePath(pi)`.
- `myMO.`**`SetRotation`**`(val)`
  With `SetRotation` one can (re)set the rotation to a specific angle. The difference between `myMO.SetRotation(val)` and `myMO.Rotation = val` is that the former sets the rotation to `val`, while the latter *increases* the current rotation angle by the value specified in `val`.
- `myMO.`**`Smoothloop`**`()`
  `Smoothloop` may help in smoothing the transition when looping an action. Sometimes the looping of the movie is not perfect (the transition from the last frame back to the first is not smooth). `Smoothloop` will calculate the linear distance between the position of each joint in the first and last frame, and then add an increasingly bigger offset to successive frames to completely cancel out the spatial differences by the last frame.
- `myMOs.`**`SetAll`**`('Option',val [,spec])`
  When you have initialized several Biomotions in one array, you can use this command to set an option to a certain value for all (or a selection of) Biomotions

in that array. There are three ways to use this function.

1. `myMOs.SetAll('Scramble', 1)`, will set `Scramble` to 1 for all Biomotions in `myMOs`.
2. You can also specify an array of the same length as `myMOs`, but with different values for each Biomotion object. e.g., if you have four Biomotion objects in `myMOs`, then `myMOs.SetAll('Scramble', [0 1 1 0])` will set the first and fourth to `Scramble = 0`, then the second and third to `Scramble = 1`.
3. You can also set specific items (specified in `spec`) to a specific value (specified in `val`). For example, to obtain the same results as in the second example you could call `myMOs.SetAll('Scramble', 1, [2 3])`.

- `myMOs.`**`SetAllVect`**`('Option', val [, spec])`
  Works as `SetAll`, but `val` is a vector. Therefore, to use option 2 and 3 as explained for `SetAll`, make a 2-D array with the rows containing the different vectors for the different actions. e.g., for option 3, type `myMOs.SetAllVect('position3D', [200 0 0; 300 0 0], [2 4])`, which will offset the second item by `[200 0 0]` and the fourth item by `[300 0 0]`.
- `myMO.`**`SaveData3D`**`('outputname')`
  Saves the `JointsInfo` data in the Biomotion class in the format "data3d." It is the default data that the Biomotion class accepts. You can afterwards make a new Biomotion object with this data: `newBM = BioMotion('outputname')`. This save option may be useful when you have created a new action (e.g., by morphing two actions, or otherwise).
- `myMO.`**`ScramblePositions`**`()`
  `ScramblePositions` will rescramble the marker locations based on the values of `ScrambleWidth`, `ScrambleHeight`, and `ScrambleDepth`. These values need to be set before this function is called in order to use custom values for the bounding box for scrambling. If you do not set these values first, then these values are set automatically, and the call will be superfluous.
- `myMO.`**`SetLimitedLifeParameters`**`()`
  By default the settings for limited lifetime stimuli will be: `maxlife = 1, ndots = nPointLights, and type = 'async'`. If no skeleton is provided, the individual joints will be used (hence the default for ndots). Note, that this will not lead to a limited lifetime stimulus on the screen, because each joint has a lifetime of 1, but will also be immediately redrawn (because by default `ndots` equals the number of limb segments, which is the number of joints in this case). To get a real limited lifetime stimulus one will have to provide a skeleton (so dots can be drawn along the limb segments), or decrease the number of displayed dots (in this case the dots will have a limited lifetime, and will be drawn on the joints). To change the settings for limited lifetime stimuli `myMO.SetLimitedLifeParameters` can be used as follows. To change, e.g., the number of dots to four per frame, and the lifetime to six frames: `myMO.SetLimitedLifeParameters('ndots', 4, 'maxlife', 6);`

Appendix II: Simple programs

First example

This program will generate a biological motion action, and displays it using MATLAB and PsychToolbox. We assume here that you have a folder in the current directory called "actions" that contains all your motion-capture files. We also assume that one of those files is called 60_06.txt, which, even though it has the extension txt, is really a bvh file.

```
clear all;                              % clear current workspace

addpath '/path_to_BiomotionToobox/BiomotionToolbox' % add path of %BioMotion class

movementdir = './actions/';             % source folder of the motion files

bmfile =  '60_06.txt';                  % name of the action file
combiname = [movementdir bmfile];       % exact file location

win = SetScreen('OpenGL',1);            % Open a screen (** see note below)
SetProjection(win);                     % Set the type of projection (** see note below)

try
    bm = BioMotion(combiname,'Filetype','bvh'); % load the data, tell the function
%that you have a bvh file even though the extension is txt
    bm.Loop = 1;                % Set the parameter to Loop

    for fr = 1:bm.nFrames           % Run through frames 1 until the end
        if KbCheck                  % Check for button presses, exit if any
            break;
        end
        moglDrawDots3D(win.Number,bm.GetFrame(fr),7); % draw the dots, note the call
%to GetFrame
        Screen('Flip',win.Number);  %display the dots on the screen
    end
    clear screen;
catch ME
    clear screen;
    rethrow(ME);
end
```

Code for Example 1. Note that the functions `SetScreen` and `SetProjection` are not standard MATLAB functions but can be downloaded at www.jeroenvanboxtel.com/software.

Note that we called `bm=BioMotion(combiname,'Filetype','bvh');` with the property Filetype set to bvh. This was done because 60_06.txt is not in data3d format, nor in vanrie format (the two formats using txt), but in bvh format.

Now, if we decide that we want to scramble the figure we just add one line after `bm.Loop=1`, namely:

`bm.Scramble=1;`

Similarly, if we want to invert the action we add:

`bm.Invert=1;`

Selecting joints to display

If we want to display only the following joints: [17 19 20 21 26 27 28 3 4 5 8 9 10], we can do that in two ways. We can change the initialization (which will make the for-loop faster, but means that only those joint will be available after initialization, and the selection cannot be expanded):

`bm=BioMotion('60_06.txt','Filetype','bvh','Selectjoints',[17 19 20 21 26 27 28 3 4 5 8 9 10]);`

Alternatively, we only ask for certain joints when we draw the dots in the buffer (this is slightly slower, but it allows you to select other joints in another call to `GetFrame`). Replace `bm.GetFrame(fr)` with:

`bm.GetFrame(fr,[17 19 20 21 26 27 28 3 4 5 8 9 10])`

Second example

Now we want to have a display with two identical actions, one on each side of the display. Each should also rotate in depth, and one of them is scrambled. (Such a display could be used in developmental psychology research, in a preferential looking task.)

```
clear all;                              % clear current workspace

addpath '/path_to_BiomotionToobox/BiomotionToolbox' % add path of BioMotion class

movementdir = './actions/';             % source folder of the motion files

bmfile =  '60_06.txt';                  % name of the action file
combiname = [movementdir bmfile];       % exact file location

win = SetScreen('OpenGL',1);            % Open a screen (** see note below)
SetProjection(win);                     % Set the type of projection (** see note below)

try
    bm1 = BioMotion(combiname,'Filetype','bvh');% load first figure
    bm2 = BioMotion(combiname,'Filetype','bvh');% load second figure

    bmarray = [bm1 bm2];
    SetAll(bmarray,'Loop',1);   % Set the parameter to Loop

    %%Or you can do:
    % bm1.Loop = 1;             % Set the parameter to Loop
    % bm2.Loop = 1;
    %%Or alternatively
    % bmarray(1).Scramble = 1;
    % bmarray(2).Scramble = 1;

    bmarray(2).Scramble = 1;        % Scramble the second actor

    %%Put the actions in different positions
    SetAllVect(bmarray,'Position3D',[-win.Width/4 0 0 ; win.Width/4 0 0]);
    %%Or you can do:
    % bm1.Position3D = [-win.Width/4 0 0];
    % bm2.Position3D = [win.Width/4 0 0];

    for fr = 1:1000                 % Run through frames 1 until frame 1000
        if KbCheck                  % Check for button presses, exit if any
            break;
        end
        SetAll(bmarray, 'Rotation',pi/180); % Rotate actions 1 deg/frame
        % draw the dots, note that there is only one call to GetFrame
        moglDrawDots3D(win.Number,bmarray.GetFrame(fr),7);

        Screen('Flip',win.Number);  %display the dots on the screen
    end
    clear screen;
catch ME
    clear screen;
    rethrow(ME);
end
```

Code for Example 2. Note that the functions `SetScreen` and `SetProjection` are not standard MATLAB functions but can be downloaded at www.jeroenvanboxtel.com/software.

Custom scrambling

Spatial scrambling

If the user wants to change the default dimensions of the scrambling, the user can set the dimensions manually before setting `Scramble = 1`, e.g.:

```
bmarray(2).ScrambleWidth = 50;
bmarray(2).ScrambleHeight = 100;
bmarray(2).ScrambleDepth = 50;
bmarray(2).Scramble = 1;
```

If the dimensions are changed after `ScrambleWidth`, `ScrambleHeight`, and `ScrambleDepth` are set, `Scramble` will have to be set back to 0 and then to 1 (in order to rescramble, and for the new dimensions to take effect). Alternatively, you can call `ScramblePositions` after setting `ScrambleWidth`, etc:

```
bmarray(2).ScramblePositions;
```

It is also possible to input the desired scrambling amounts. This can be done by calling `bmarray(2).ScrambleOffsets = arraywithpositions`. The positions in `arraywithpositions` are the 3-D starting coordinates, relative to [0 0 0]. In order to set `ScrambleOffsets` it is important to set `Scramble` to 1, *before* setting `ScrambleOffsets`, otherwise the input is overwritten by the default scrambling function:

```
bmarray(2).Scramble = 1;
```

```
bmarray(2).ScrambleOffsets = arraywithpositions;
```

Phase Scrambling

Phase scrambling can also be set manually. The user will need to define `PhaseOffsets`. In order for it to take effect the user needs to set `PhaseScramble = 1` *after* `PhaseOffsets` is set, e.g.:

```
bmarray(1).PhaseOffsets = [100 100 100 200 200 200 400];
bmarray(1).PhaseScramble = 1;
```

If `PhaseScramble` was already set to 1 before setting `PhaseOffsets`, the user can call `bmarray(1).PhaseScramble = 3` afterwards. This call will overwrite the old phase scrambling and use the new user-defined values.

```
bmarray(1).PhaseScramble = 1;
[...other code...]
bmarray(1).PhaseOffsets = [100 100 100 200 200 200 400];
bmarray(1).PhaseScramble = 3;
```

Third example

In this example we display two identical actions. One of them is a limited lifetime stimulus, with the dots drawn along a user-provided skeleton. The skeleton for the limited lifetime action in this example is defined according to the drawing in Figure A1. The example also shows a limited lifetime stimulus in which the dots are redrawn on the joints. This happens when no skeleton is provided.

```
try
    skel = [2 3; 3 4; 5 6; 6 7; 8 9; 9 10; 11 12; 12 13];

    bm = BioMotion('./myactiondir/myaction.txt');
    bm.Loop = 1;
    bm.Rotation = pi/2;

    bm2 = Copy(bm);
    bm2.Position3D = [200 0 0];

    % Limited lifetime, dots drawn along limb segments (c.f. Beintema &
    % Lappe (2002)
    bm.Skeleton = skel;
    bm.SetLimitedLifeParameters('ndots',5,'maxlife',2);

    % Limited lifetime, dots drawn on joints only. Need to decrease 'ndots'
    % to <nPointLights, in order to see the effect
    bm2.SetLimitedLifeParameters('ndots',4,'maxlife',2);

    bmarray = [bm bm2];
    SetAll(bmarray,'LimitedLife',1);

    win = SetScreen('OpenGL',1);
    SetProjection(win);

    for fr  = 1:500
        while KbCheck
        % Press any key to inspect the stimulus. E.g. are the nbmr of dots correct?
        end
        moglDrawDots3D(win.Number,bmarray.GetFrame(fr),7,[255 0 0],[],2);
        Screen('Flip',win.Number);
    end
    clear screen

catch ME
    clear screen
    rethrow(ME);
end
```

Code for Example 3. Note that the functions `SetScreen` and `SetProjection` are not standard MatLab functions but can be downloaded at www.jeroenvanboxtel.com/software.

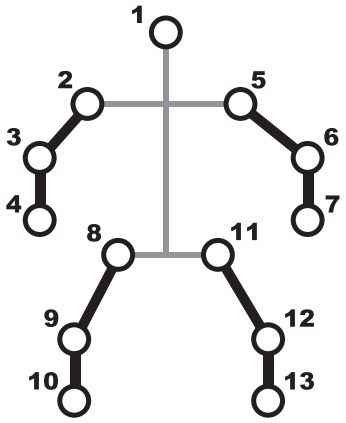

Figure A1. In this example we assume a 13-jointed action. The joints are in the order as defined in the figure. In this example we will draw markers along the limb segments indicated by the thick black lines.

Appendix III: Motion capture databases

Here we provide several Motion capture databases that can be accessed online. See http://www.jeroenvanboxtel.com/MocapDatabases.html for an updated list:

- Pollick lab Body Movement Library (Ma et al., 2006) http://paco.psy.gla.ac.uk/index.php?option=com_jdownloads&view=viewcategories&Itemid=62
- Leuven Action Database (Vanrie & Verfaillie, 2004) http://ppw.kuleuven.be/english/lep/resources/action
- Carnegie Mellon Mocap Database http://mocap.cs.cmu.edu
- Human Motion Database (Guerra-Filho & Biswas 2012) http://smile.uta.edu/hmd/hmd.htm
- OptiTrack Samples http://www.naturalpoint.com/optitrack/downloads/
- ICS Action Database http://www.ics.t.u-tokyo.ac.jp/action/
- HumanEva http://vision.cs.brown.edu/humaneva/
- Motion Capture Database HDM05 (Müller et al., 2007) http://www.mpi-inf.mpg.de/resources/HDM05/
- Mocapdata http://www.mocapdata.com/
- Human Identification at a Distance http://www.cc.gatech.edu/cpl/projects/hid/
- ACCAD database http://accad.osu.edu/research/mocap/mocap_data.htm
- Movlab http://movlab.ulusofona.pt/cms/index.php?option=com_content&view=article&id=160&Itemid=104&lang=en