KRAJJAT documentation

For version 2.0

Last edited: 14/05/2023

Contents

Classes

Sequence

Description

Default class for motion sequences in the toolbox. An instance of this class will be one motion sequence. The class contains several methods to perform **pre-processing** or displaying **summarized data** in text form (see public methods).

Initialisation

```
Sequence(path=None, path_audio=None, name=None, convert_to_seconds="Auto") -> Sequence
```

Short description

Creates an instance from the class Sequence.

Full description

Creates an instance from the class Sequence and returns a Sequence object, allowing to be manipulated for processing or displaying. Upon creation, the function tries to assign a name to it based on the provided name parameter or path parameter. It then proceeds to load the sequence if a path has been provided.

Parameters

Parameter	Type	Requirement	Default value
path	str	Optional	*None*

Path to the motion sequence. The path should be the absolute path (starting from the root or the drive, e.g. C:/Users/Elliot/Documents/Recordings). The path should point to a folder or a single file:

- If the path points to a **folder**, the folder should contain individual files (.csv, .json, .tsv, .txt or .xlsx) for each pose of the sequence. Moreover, the alphabetical order of the files must also follow the chronological order of the poses they contain.
- If the path points to a **file**, the file should contain all the poses, and end with one of the following extensions: .csv, .json, .tsv, .txt, .xlsx.

Regarding the content of the files:

- For .csv (comma-separated values), .xlsx (Microsoft Excel files), .tsv (tabulation-separated values) or .txt (text) files, the first line must always be the labels, with the first being "Timestamp" and the subsequent labels being in group of three columns for each joint: respectively, one for the X, Y and Z coordinates. Line 2 (in case of a single pose file) or lines 2 and on (in case of a global, sequence file) should contain the timestamps in chronological order and the X, Y and Z coordinates for each joint. As an example, a file with two joints and three poses should look like this:

Timestamp	Head_X	Head_Y	Head_Z	HandRight_X	HandRight_Y	HandRight_Z
0	0.400	0.80	1.50	1.60	2.300	4.2
0.125	0.1123	0.5813	0.2134	0.5589	0.1442	0.3337
0.250	0.2008	0.0512	0.1519	0.2018	0.1515	0.1300

  Note: global .tsv files that are output from the QualiSys software will be detected automatically and don't need any type of processing to be used.
- For .json (JavaScript object notation) single-pose files, the structure must be a dictionary with at least two entries: "Timestamp" and "Bodies". The value for "Bodies" must be a list containing a new dictionary with an entry called "Joints". The value for "Joints" must be a dictionary containing an entry for each joint, under the form of a dictionary with at least two keys: "JointType", which value is a string containing the label of the joint, and "Position". The value for "Position" must be a final dictionary containing at least the three entries "X", "Y" and "Z", with their coordinates as values. As an example, the data from the two first lines in the table above should look like this in json form:

```
{"Timestamp": 0,
 "Bodies": [
              {"Joints":
                     [
                            {"JointType": "Head",
                             "Position": {"X": 0.400, "Y": 0.80, "Z": 1.50}},
                            {"JointType": "HandRight",
                             "Position": {"X": 1.60, "Y": 2.300, "Z": 4.2}}]}]}
```

	If the file is a global file, containing multiple poses, each pose must be an element of a list, with the structure of each element being the same as described above. Note that dictionary entries other than the ones mentioned above will be ignored by the toolbox. If supplied, the initialization function will call the `Sequence._load_from_path()` function. If set on None, the sequence object remains empty with default values.		
`path_audio`	str	Optional	*None*
Path to an audio file corresponding to the sequence. The path should be an absolute path, and point to a .wav file. This path will be stored as an attribute of the Sequence object, and may be used automatically by functions using an audio file (typically, `Sequence.synchronize()` and `sequence_reader`).			
`name`	str	Optional	*None*
Defines a name for the Sequence instance. If a string is provided, the attribute ***name*** will take its value. If not, see `Sequence._define_name_init()`.			
`time_unit`	str or bool	Optional	`"auto"`
If set on `"auto"`, the function `Sequence._calculate_relative_timestamps()` will automatically detect if to divide the timestamps by 10 000 000, 1 000, or no. If set on `"100ns"`, divides the timestamps from the file by 10 000 000. Typically, this is due to the output timestamps from Kinect being in tenth of microsecond (C# system time). If set on `"ms"`, divides the timestamps from the file by 1 000. If set on `"s"`, the function will preserve the timestamps as they are in the file. The parameter also allows other units. See the documentation for the function `Sequence._calculate_relative_timestamps()`.			
`start_timestamps_at_zero`	bool	Optional	*False*
If set on *True*, the timestamp of the first pose of the sequence will be set at 0, and the timestamps of the other poses will be reassigned to keep the same delay from the first pose. As such, the attributes `timestamp` and `relative_timestamp` from every pose will be equal.			
`verbose`	int	Optional	1
Sets how much feedback the code will provide in the console output: • `0`: **Silent mode**. The code won't provide any feedback, apart from error messages. • `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Attributes created

Attribute	Type	Value upon initialization
`path`	str	*path*
Path to the sequence passed as a parameter.		
`path_audio`	str	*path_audio*
Path to the audio file passed as a parameter.		
`name`	str	*name*
Name of the sequence passed as a parameter.		
`files`	list	[]
List of files contained in the path. The list will be of size 1 if the path points to a single file.		
`poses`	list	[]
List of all the poses objects of the sequence.		
`randomized`	bool	*False*
Testifies if the starting position of the joints have been randomized.		
`date_recording`	datetime	*None*
The date at which the recording was performed, extracted from the file.		

Subfunctions called

- `Sequence._define_name_init()`
- `Sequence._load_from_path()`

Hierarchy

- `Sequence.__init__()`

- Sequence._define_name_init()
- Sequence._load_from_path()
  - Sequence._fetch_files_from_folder()
  - Sequence._load_poses()
    - Sequence._load_single_pose_file()
      - Sequence._load_date_recording()
      - Sequence._create_pose()
    - Sequence._load_sequence_file()
      - Sequence._load_date_recording()
      - Sequence._create_pose()
  - Sequence._calculate_relative_timestamps()
  - Sequence._calculate_velocities()

Magic methods

Sequences are defined as immutable items, by acting indirectly on the `poses` attribute. As such, it is possible to get the length of a Sequence instance, and to use indexing.

Length

```
len(sequence) -> int
```

Returns the number of poses in the sequence (i.e., the length of the attribute `poses`).

Indexing

```
sequence[key] -> Pose
```

Returns the pose or poses of index specified by the parameter *key*.

Representation

```
repr(sequence) -> Pose
```

Returns the *name* attribute of the sequence.

Public methods

Sequence.set_name()

```
sequence.set_name(name) -> None
```

Short description

Sets the `name` attribute of the Sequence instance.

Full description

Sets the `name` attribute of the Sequence instance. This name can be used as display functions or as a means to identify the sequence (typically, `joint_temporal_plotter()`).

Parameters

Parameter	Type	Requirement	Default value
`name`	str	Required	–
The name to give to the sequence.			

Sequence.set_path_audio()

`sequence.set_path_audio(path_audio) -> None`

Short description

Sets the `path_audio` attribute of the Sequence instance.

Full description

Sets the `path_audio` attribute of the Sequence instance. This path may be used automatically by functions using an audio file (typically, `Sequence.synchronize()` and `sequence_reader`).

Parameters

Parameter	Type	Requirement	Default value
`path_audio`	str	Required	–
The name to give to the sequence.			

Sequence.set_first_timestamp()

```
sequence.set_first_timestamp(first_timestamp) -> None
```

Short description

Attributes a new timestamp to the first pose of the sequence and delays the timestamps of the other poses accordingly.

Full description

This function allows to define a new starting timestamp for the poses of the sequence. All of the timestamps of the sequence will be modified, but the relative timestamps will be left untouched.

Parameters

Parameter	Type	Requirement	Default value
first_timestamp	float	Required	–
The timestamp to attribute to the first pose of the sequence (in seconds).			

Sequence.get_pose()

```
sequence.get_pose(pose_index) -> Pose
```

Short description

Returns the pose instance corresponding to the index passed as parameter.

Full description

Returns the pose instance corresponding to the index passed as parameter.

Parameters

Parameter	Type	Requirement	Default value
pose_index	int	Required	–
The index of the pose.			

Outputs

Parameter	Type
pose	Pose
A pose from the sequence.	

Sequence.get_number_of_poses()

```
sequence.get_number_of_poses() -> int
```

Short description

Returns the number of poses in the sequence.

Full description

Returns the number of poses in the sequence.

Outputs

Parameter	Type
number_of_poses	int
Number of poses in the sequence.	

Sequence.get_date_recording()

```
sequence.get_date_recording() -> datetime
```

Short description

Returns the date and time of the recording as a datetime object, if it exists.

Full description

This function returns a datetime object corresponding to the date at which the recording was started. If the date of the recording wasn't specified in the original file, the value will be *None*.

Outputs

Parameter	Type
date	datetime
A datetime instance corresponding to the date at which the recording was started.	

Sequence.get_printable_date_recording()

```
sequence.get_printable_date_recording() -> str
```

Short description

Returns the date and time of the recording as a string, if it exists.

Full description

This function returns a string corresponding to the date at which the recording was started, with the weekday, day, month (in letters), year, hour, minutes and seconds (e.g. `"Wednesday 21 October 2015, 07:28:00"`). If the attribute `date_recording` of the sequence is *None*, the value returned is `"No date found"`.

Outputs

Parameter	Type
`date`	str
A formatted string of the date of the recording (e.g. `"Wednesday 21 October 2015, 07:28:00"`), or `"No date found"`	

Sequence.get_timestamps()

```
sequence.get_timestamps(relative=True) -> list
```

Short description

Returns a list of the timestamps for every pose, in seconds.

Full description

Returns a list of the timestamps for every pose, in seconds. By default, the timestamps returned are relative to the first pose; as such, the list should start with the value 0. It is possible to return the original timestamps of the sequence by setting the parameter `relative` on `False`.

Parameters

Parameter	Type	Requirement	Default value
`relative`	bool	Optional	`True`
Defines if the returned timestamps are relative to the first pose (in that case, the timestamp of the first pose will be 0), or the original timestamps.			

Outputs

Parameter	Type
`timestamps`	list
List of the timestamps of all the poses of the sequence, in seconds.	

Sequence.get_time_between_two_poses()

`sequence.get_time_between_two_poses(pose1, pose2) -> float`

Short description

Returns the difference between the timestamps of two poses, in seconds.

Full description

Returns the difference between the timestamps of two poses, in seconds.

Parameters

Parameter	Type	Requirement	Default value
`pose1`	int	Required	–
The index of the first pose.			
`pose2`	int	Required	–
The index of the second pose.			

Outputs

Parameter	Type
`time`	float
Time elapsed between the two poses.	

Uses

- `Sequence.correct_jitter()`

Sequence.get_duration()

```
sequence.get_duration() -> float
```

Short description

Returns the duration of the sequence, in seconds.

Full description

Returns the duration of the sequence, in seconds. The function actually returns the value of the relative timestamp of the last pose of the sequence.

Outputs

Parameter	Type
`timestamp`	int
Duration of the sequence, in seconds.	

Sequence.get_frequencies()

```
sequence.get_frequencies() -> list, list
```

Short description

Returns a list of the frequencies of poses per second and a list of the matching timestamps.

Full description

This function calculates, for each pose, the inverse of the time elapsed since the previous pose (in seconds), in order to obtain a list of frequencies ("framerates") across time. It returns the list of frequencies, and the corresponding timestamps (starting on the second pose).

Outputs

Parameter	Type
`frequencies`	list
Framerates for all the poses of the sequence, starting on the second pose.	
`time_points`	list
Timestamps of the sequence, starting on the second pose.	

Sequence.get_average_frequency()

```
sequence.get_average_frequency() -> float
```

Short description

Returns the average number of poses per second of the sequence.

Full description

This function calculates the frequency of poses for each pose and divides it by the total number of poses minus one. The result is an average number of poses per second (framerate) for the sequence.

Outputs

Parameter	Type
`average_frequency`	float
Average number of poses per second for the sequence.	

Sequence.get_min_frequency()

```
sequence.get_min_frequency() -> float
```

Short description

Returns the minimum frequency of poses per second of the sequence.

Full description

This function returns the minimum value for the frequencies of poses per second in the whole sequence, which is equal to 1 over the maximum time between two poses in the sequence.

Outputs

Parameter	Type
`min_frequency`	float
Minimum number of poses per second for the sequence.	

Sequence.get_max_frequency()

```
sequence.get_max_frequency() -> float
```

Short description

Returns the maximum frequency of poses per second of the sequence.

Full description

This function returns the maximum value for the frequencies of poses per second in the whole sequence, which is equal to 1 over the minimum time between two poses in the sequence.

Outputs

Parameter	Type
max_frequency	float
Maximum number of poses per second for the sequence.	

Sequence.get_joint_coordinate_as_list()

`sequence.get_joint_as_list(joint_label, axis) -> list`

Short description

Returns a list of all of the values for one specified coordinate of a specified joint label.

Full description

This function returns a list of the values for a *joint_label* across time, for one specific *axis* (x, y or z).

Parameters

Parameter	Type	Requirement	Default value
joint_label	string	Required	–
The label of the joint (e.g. "Head").			
axis	string	Required	–
The required axis ("x", "y" or "z").			

Outputs

Parameter	Type
values	list
A list of the values on one axis for the specified joint.	

Sequence.get_joint_velocity_as_list()

```
sequence.get_joint_velocity_as_list(joint_label) -> list
```

Short description

Returns a list of all of the velocities (distance travelled over time) for a specified joint.

Full description

Returns a list of all of the velocities (distance travelled over time) for a specified joint.

Parameters

Parameter	Type	Requirement	Default value
joint_label	string	Required	–
The label of the joint (e.g. "Head").			

Outputs

Parameter	Type
values	list
A list of the velocities for the specified joint.	

Sequence.get_velocities()

`sequence.get_velocities() -> OrderedDict`

Short description

Returns a dictionary containing a list of velocities (distance travelled over time between two poses) for each joint.

Full description

This function creates a dictionary with the joint labels as keys, and lists of velocities as values. Velocities are calculated using the `calculate_velocity()` function, and are defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocities are returned in meters per second.

Outputs

Parameter	Type
`dict_velocities`	OrderedDict
Dictionary with joint labels as keys, and lists of velocity as values (in meters per second), ordered chronologically.	

Subfunctions called

- `Sequence.get_joint_velocity_as_list()`

Uses

- `Sequence.get_max_velocity_whole_sequence()`
- `velocity_plotter()`

Sequence.get_max_velocity_whole_sequence()

`sequence.get_max_velocity_whole_sequence() -> float`

Short description

Returns the single maximum value of the velocity across every joint of the sequence.

Full description

This function calculates the single maximum velocity for across every joint of the whole sequence. The velocities are first calculated using the `Sequence.get_velocities()` function. The velocity of a joint is defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocity is returned in meters per second.

Outputs

Parameter	Type
`max_velocity_whole_sequence`	float
Maximum value of the velocity across every joint of the sequence, in meters per second.	

Uses

- `velocity_plotter()`

Sequence.get_max_velocity_single_joint()

`sequence.get_max_velocity_single_joint(joint_label) -> float`

Short description

Returns the maximum value of the velocity for a given joint.

Full description

This function calculates the maximum velocity for a given joint across the whole sequence. The velocities are first calculated using the `Sequence.get_velocities()` function. The velocity of a joint is defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocity is returned in meters per second.

Parameters

Parameter	Type	Requirement	Default value
`joint_label`	str	Required	–
The label of the joint.			

Outputs

Parameter	Type
`max_velocity`	float
Maximum value of the velocity for a given joint, in meters per second.	

Sequence.get_max_velocity_per_joint()

`sequence.get_max_velocity_per_joint() -> OrderedDict`

Short description

Returns the maximum value of the velocity for each joint of the sequence.

Full description

This function calculates the maximum velocity for every joint across the whole sequence. The velocities are first calculated using the `Sequence.get_velocities()` function. The velocity of a joint is defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocity is returned in meters per second.

Outputs

Parameter	Type
`max_velocity_per_joint`	OrderedDict
Dictionary with joint labels as keys, and maximum velocity for the joint as values, in meters per second.	

Sequence.get_total_velocity_whole_sequence()

```
sequence.get_total_velocity_whole_sequence() -> float
```

Short description

Returns the sum of the velocities of every joint across all the poses of the sequence.

Full description

This function calculates the sum of the velocities of each joint across all poses, in order to provide a value representative of the global "quantity of movement" produced during the sequence. The velocities are first calculated using the `Sequence.get_velocities()` function. The velocity of a joint is defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocity is returned in meters per second.

Outputs

Parameter	Type
`total_velocity_whole_sequence`	float
Sum of the velocities across every joint and poses of the sequence, in meters per second.	

Sequence.get_total_velocity_single_joint()

`sequence.get_total_velocity_single_joint(joint_label) -> float`

Short description

Returns the sum of the velocities for a given joint.

Full description

This function calculates the sum of the velocities for a given joint across all poses. The velocities are first calculated using the `Sequence.get_velocities()` function. The velocity of a joint is defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocity is returned in meters per second.

Parameters

Parameter	Type	Requirement	Default value
`joint_label`	str	Required	–
The label of the joint.			

Outputs

Parameter	Type
`total_velocity_single_joint`	float
Sum of the velocities across all poses for a single joint, in meters per second.	

Uses

- `Sequence.get_total_velocity_per_joint()`
- `Sequence.get_stats()`

Sequence.get_total_velocity_per_joint()

`sequence.get_total_velocity_per_joint() -> OrderedDict`

Short description

Returns the sum of the velocities for each individual joint of the sequence.

Full description

This function calculates the sum of the velocities for each individual joint across all poses. The velocities are first calculated using the `Sequence.get_velocities()` function. The velocity of a joint is defined by the distance travelled between two poses (in meters), divided by the time elapsed between these two poses (in seconds). The velocity is returned in meters per second.

Outputs

Parameter	Type
`total_velocity_per_joint`	OrderedDict
Dictionary with joint labels as keys, and sum of the velocities across all poses as values, in meters per second.	

Uses

- `velocity_plotter()`

Sequence.get_subject_height()

```
sequence.get_subject_height(verbose=1) -> float
```

Short description

Returns an estimation of the height of the subject, in meters.

Full description

Returns an estimation of the height of the subject, in meters, based on the successive distances between a series of joints. For each pose, the distance between a series of joints, two by two, is calculated. Then, the average height between all the poses is calculated and returned. Some of these joints are imaginary, based on the average of the position of two or four joints. For the Kinect system, the joints used are Head, Neck, SpineShoulder, SpineMid, SpineBase, the average between KneeRight and KneeLeft, the average between AnkleRight and AnkleLeft, and the average between FootRight and FootLeft. For the Qualisys system, the joints used are Head Top, the average between ShoulderTopRight and ShoulderTopLeft, Chest, the average between WaistBackRight, WaistBackLeft, WaistFrontRight and WaistFrontLeft, the average between KneeRight and KneeLeft, the average between AnkleRight and AnkleLeft, and the average between ForefootOutRight and ForefootOutLeft.

Parameters

Parameter	Type	Requirement	Default value
verbose	int	Optional	1

Sets how much feedback the code will provide in the console output:

- 0: **Silent mode**. The code won't provide any feedback, apart from error messages.
- 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps.
- 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.

Outputs

Parameter	Type
height	float

The estimated height of the subject, in meters.

Uses

- Sequence.get_stats()

Sequence.get_subject_arm_length()

```
sequence.get_subject_arm_length(side="left", verbose=1) -> float
```

Short description

Returns an estimation of the length of the left or right arm of the subject, in meters.

Full description

Returns an estimation of the length of the arm (left or right, defined by the parameter *side*) of the subject, in meters, based on the successive distances between a series of joints. For each pose, the distance between a series of joints, two by two, is calculated. Then, the average height between all the poses is calculated and returned. For the Kinect system, the joints used are (Left/Right)Shoulder, (Left/Right)Elbow, (Left/Right)Wrist and (Left/Right)Hand. For the Qualisys system, the joints used are (Left/Right)ShoulderTop, (Left/Right)Arm, (Left/Right)Elbow, (Left/Right)WristOut and (Left/Right)HandOut.

Parameters

Parameter	Type	Requirement	Default value
side	string	Optional	"left"
Side of the arm you want the measure from. Can be "left" or "right".			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
arm_length	float
The estimated arm length of the subject, in meters.	

Uses

- Sequence.get_stats()

Sequence.get_stats()

```
sequence.get_stats(tabled=False) -> OrderedDict or list
```

Short description

Returns a dictionary containing a series of statistics regarding the sequence.

Full description

Returns an OrderedDict (if the parameter *tabled* is set on *False*) or a two-dimensional list (if the parameter *tabled* is set on *True*) containing diverse statistics about the sequence. See the full list of keys in the output section.

Parameters

Parameter	Type	Requirement	Default value
tabled	bool	Optional	False
Returns the statistics under the form of an OrderedDict if set on *False*, or a two-dimensional list if set on *True*. The latter can be used to assemble the statistics of many sequences under one big table, for example.			

Outputs

Parameter	Type
stats	OrderedDict or list

In the case where tabled was set on False, stats will be an OrderedDict with the following keys:

- "Path": The path attribute of the sequence.
- "Date of recording": Output of **Sequence**.get_printable_date_recording().
- "Duration": Output of **Sequence**.get_duration() (seconds).
- "Number of poses": Output of **Sequence**.get_number_of_poses().
- "Subject height": Output of **Sequence**.get_subject_height() (meters).
- "Left arm length": Output of **Sequence**.get_subject_arm_length("left") (meters).
- "Right arm length": Output of **Sequence**.get_subject_arm_length("right") (meters).
- "Average frequency": Output of **Sequence**.get_average_frequency().
- "SD frequency": Standard deviation of the frequency of the sequence.
- "Min frequency": Output of **Sequence**.get_min_frequency().
- "Max frequency": Output of **Sequence**.get_max_frequency().
- "Average velocity X": Output of **Sequence**.get_total_velocity_single_joint(X) divided by the total number of poses. This key has one entry per joint label.

Subfunctions called

- Sequence.get_duration()
- Sequence.get_printable_date_recording()
- Sequence.get_number_of_poses()
- Sequence.get_subject_height()
- Sequence.get_subject_arm_length()
- Sequence.get_average_frequency()
- Sequence.get_min_frequency()
- Sequence.get_max_frequency()
- Sequence.get_total_velocity_single_joint()

Sequence.correct_jitter()

```
sequence.correct_jitter(velocity_threshold, window, window_unit="pose",
method="default", name=None, verbose=1) -> Sequence
```

Short description

Detects and corrects rapid twitches and jumps in a motion sequence.

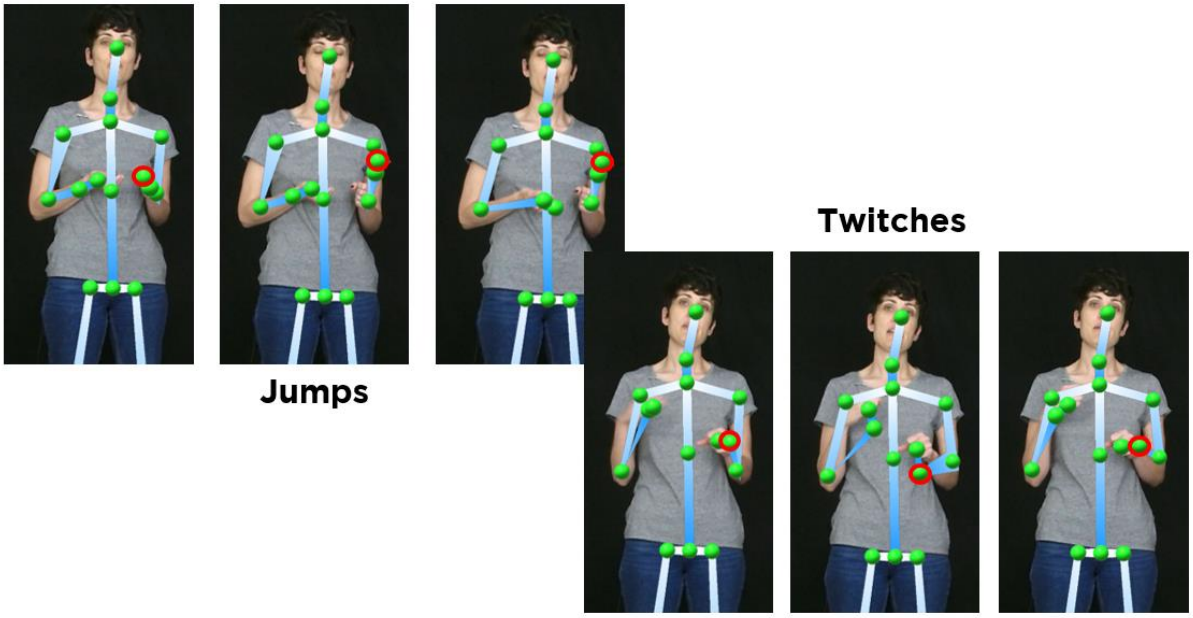
Full description

This function allows to correct the rapid movements of the joints from a recording, typically due to poor automatic detection of a joint in space, and resulting in aberrant, unrealistic displacements from the joints. Though originally developed to handle Kinect data, the algorithm can be used for any 3D-tracked biological movement data having timestamps and X, Y and Z coordinates for each pose. The *velocity_threshold* parameter defines the threshold of velocity, in meters per second, over which a movement is considered as aberrant; the *window* parameter, defined in individual poses or in milliseconds (via the parameter *window_unit*), defines the maximum duration of an aberrant movement – if the duration exceeds the window, the joint is corrected and smoothed out for all the poses of the window, using the defined *method* (linear by default).

Principle

This algorithm has been developed to handle fast, non-biological movements on the superior limbs, i.e., the hands and arms. Two types of aberrant movements have been defined: jumps and twitches.

- **Jumps** are defined as unrealistic displacements of the position of a joint in between frames, **without a subsequent return** at the original **position** under a few frames. In the example below, the tracked joint for the right hand suddenly moves between frames 1 and 2, and remains in this position in frame 3.
- **Twitches** are defined as unrealistic displacements of the position of a joint in between frames, **with a subsequent return** at the original position under a few frames. For instance, in the image below, the right wrist joint suddenly moves between frames 1 and 2, but comes back to the original position on frame 3.



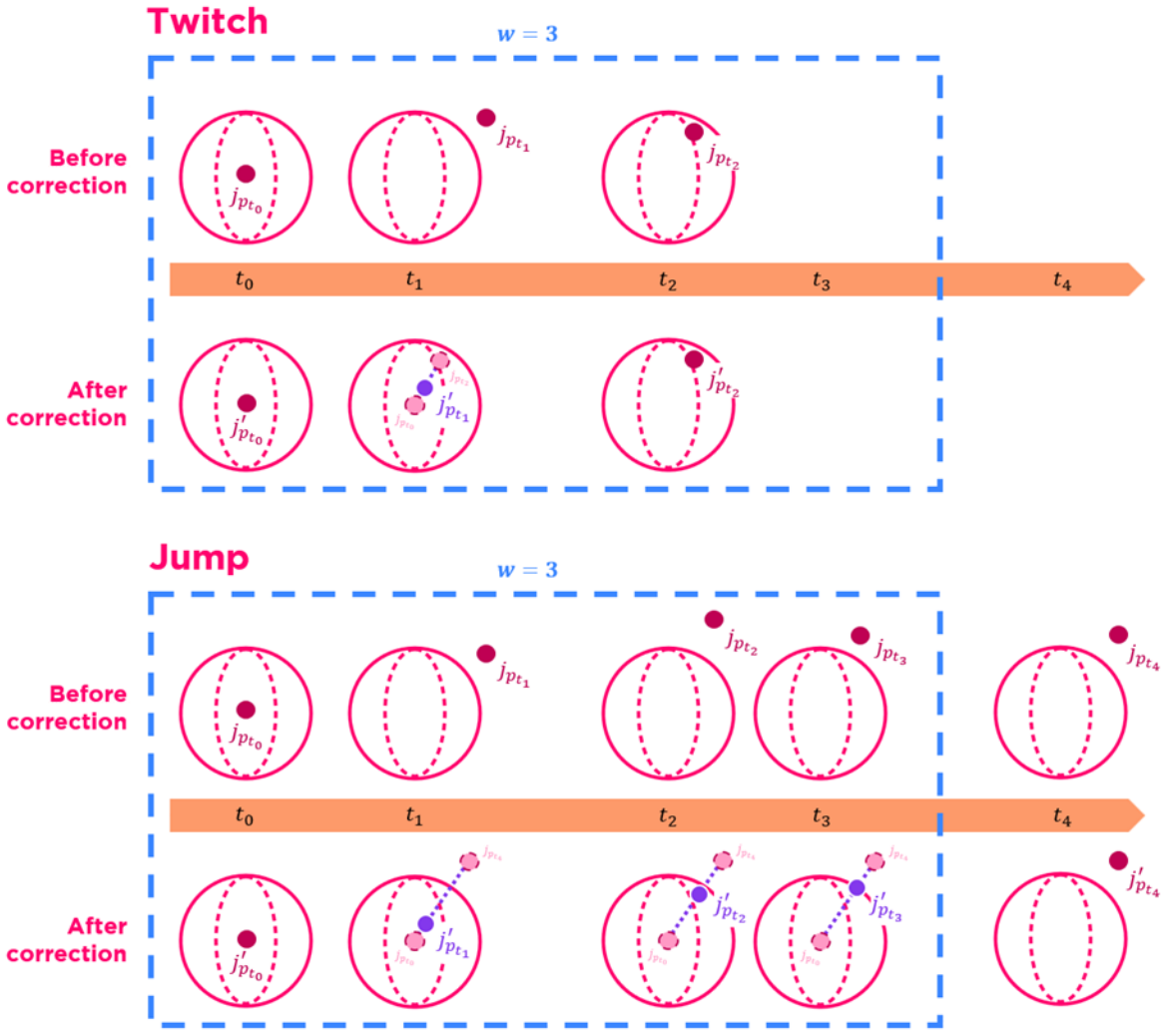
The function takes two parameters in input: the velocity threshold and the window.

- The **velocity threshold** is the amount of movement between two poses over which we consider a displacement to be unrealistic. It is defined in metres per second.

- The **window** is the amount of time (or poses) defining if an over-threshold displacement is a jump or a twitch. It can be defined in milliseconds, in second or in number of poses.

For every joint in every pose, a velocity is calculated, in metres per second. If this velocity is over the defined threshold, the algorithm detects if the aberrant movement is a jump or a twitch. If the velocity (defined by the distance travelled between the last pose *p* before the aberrant movement, and a pose within the window p+n, over the time between poses p and p+1) for a pose in the window comes back below threshold, the movement is considered as a twitch, and joint positions of the poses containing an aberrant movement are corrected using the last pose before the aberrant movement and the first pose below threshold after the aberrant movement as reference. If, for every pose of the window, the velocity does not come back below threshold, it is considered as a jump. In order to smooth out the movement of the jump, the last pose before the aberrant movement and the last pose of the window are used as reference for correction.

Below can be found visual representations of what a twitch and a jump correction look like, in the case where poses have irregular timestamps and for a linear correction (default method).



If expressed as an inequation, the velocity threshold can be expressed as follows:

Let's call j_{t_p} a joint, as being a 3-dimensional (x, y, and z, in meters to the origin) point being located in space at a moment t_p (in seconds), where p iterates through all the poses of a video. For a velocity threshold s and a window w, if the following inequation is true:

$$(\mathbf{1})\quad \frac{\sqrt{\left(x_{j_{t_{p+i}}} - x_{j_{t_p}}\right)^2 + \left(y_{j_{t_{p+i}}} - y_{j_{t_p}}\right)^2 + \left(z_{j_{t_{p+i}}} - z_{j_{t_p}}\right)^2}}{t_{p+1} - t_p} \geq s \text{ for } i \in [1, w] \text{ and } i \in \mathbb{N}$$

the misplacement is considered as a **jump**. If for any $i \in [0, w]$, the inequation $(\mathbf{1})$ is false, the misplacement is considered as a **twitch**.

Parameters

Parameter	Type	Requirement	Default value
`velocity_threshold`	float	Required	–
The threshold of velocity over which a movement is considered as abnormal, hence to correct. It is defined in meters per second. A good threshold must correct jumps and twitches without correcting valid, biological movement (e.g. between 0.1 and 1 m/s).			
`window`	float	Required	–
The amount of poses (by default) or time allowed for a joint to come back below threshold. If the parameter *`window_unit`* is set on poses, the window should be adjusted depending on the framerate of the recording. A window of 3 to 5 poses for a Kinect recording between 10 and 15 poses per second has been shown to give good results.			
`window_unit`	str	Optional	`"pose"`
The unit for the parameter *`window`*. • If set on `"pose"`, the parameter *`window`* will be interpreted as a number of poses. Recommended for recordings with variable framerate. • If set on `"s"` or `"ms"`, the parameter *`window`* will be interpreted as being an amount of seconds or milliseconds, respectively. The algorithm will then, on every iteration, calculate the amount of poses that has a duration the closes to the provided *`window`* value.			
`method`	str	Optional	`"default"`
The method as to how smoothen out the data of the joints detected has being part of a twitch or a jump. • If set on `"default"`, the movement is corrected linearly by taking as reference the last pose before the aberrant movement, and the first pose below threshold (in case of a twitch) or the last pose of the window (in case of a jump). • If set on `"old"`, the movement is corrected the same way as for default, but the method to check the velocity threshold is based on an old, incorrect version of the algorithm. This option is deprecated and has only been left there for version 2.0 for retro-compatibility and comparisons with old, processed files. This method will be removed in an ulterior release. • This parameter also allows for all the values accepted for the ***`kind`*** parameter in the function `scipy.interpolate.interp1d`: `"linear"`, `"nearest"`, `"nearest-up"`, `"zero"`, `"slinear"`, `"quadratic"`, `"cubic"`, `"previous"`, and `"next"`. See the documentation for this Python module for more. In case one of these values is used, all of the corrected joints are first set to (0, 0, 0) by the function, before calling the `Sequence.correct_zeros()` function to interpolate the missing values.			
`name`	bool	Optional	*`True`*
Defines the name of the output sequence. If set on *`None`*, the name will be the same as the input sequence, with the suffix `"+CJ"`.			
`verbose`	str	Optional	*`None`*
Sets how much feedback the code will provide in the console output: • `0`: **Silent mode**. The code won't provide any feedback, apart from error messages. • `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
`new_sequence`	Sequence
A new sequence having the same amount of poses and timestamps as the original, but with corrected joints coordinates.	

Subfunctions called

- Sequence._create_new_sequence_with_timestamps()
- Sequence._correct_jitter_window()
- Sequence._calculate_relative_timestamps()

Sequence.re_reference()

```
sequence.re_reference(reference_joint="auto", place_at_zero=True, name=None, verbose=1) -> Sequence
```

Short description

Changes the position of all the joints of a sequence to be relative to the position of a reference joint.

Full description

This function takes the label of a joint *reference_joint* and defines its coordinate as the origin. By default (if set on auto), this joint will be "SpineMid" for Kinect data, and "Chest" for Qualisys data. If *place_at_zero* is true, the coordinates of the reference joint will be set at (0, 0, 0) for the full duration of the sequence; if set on false, the coordinates of the joint on the first pose of the sequence will be applied for every pose. The function then calculates the new coordinates of every joint for each pose of the sequence, by keeping the distance from the reference joint. As a result, the new coordinates of the joints will not be relative to a fixed point in space, but to the position of the reference joint.

Parameters

Parameter	Type	Requirement	Default value
reference_joint_label	str	Optional	"auto"
The label of the joint to take as reference. If set on "auto", the function will try to detect the presence of, and subsequently assign "SpineMid" (Kinect data), or "Chest" (Qualisys data).			
place_at_zero	bool	Optional	*True*
If set on *True*, the positions of the joint with the label reference_joint will be set at coordinate (0, 0, 0). If set on *False*, the positions of the joint with the label reference_joint will be set at the position from the first pose of the sequence.			
name	str	Optional	*None*
Defines the name of the output sequence. If set on *None*, the name will be the same as the input sequence, with the suffix "+RF".			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
new_sequence	Sequence
A new sequence having the same amount of poses and timestamps as the original, but with re-referenced coordinates.	

Subfunctions called

- Sequence._create_new_sequence_with_timestamps()
- Sequence._calculate_relative_timestamps()

Sequence.trim()

```
sequence.trim(start=None, end=None, use_relative_timestamps=True, name=None,
verbose=1, add_tabs=0) -> Sequence
```

Short description

Trims a sequence according to a starting timestamp (by default the beginning of the original sequence) and an ending timestamp (by default the end of the original sequence). Timestamps must be provided in seconds.

Full description

This function returns a Sequence instance containing a subset of the poses from the original sequence. The preserved poses have a timestamp superior to *start* seconds (if *start* is *None*, the value of the timestamp or relative timestamp of the first pose of the sequence will be assigned to it) and inferior to *end* seconds (if *end* is *None*, the value of the timestamp or relative timestamp of the last pose of the sequence will be assigned to it). If *use_relative_timestamps* is set to *True*, the values of *start* and *end* must be provided considering that the first pose of the sequence has 0 has a relative timestamp; if *use_relative_timestamps* is set to *False*, the values of *start* and *end* must be provided considering the original timestamps of the poses.

Parameters

Parameter	Type	Requirement	Default value
start	float	Optional	*None*
The timestamp over which the poses will be preserved. If set on *None*, the beginning of the sequence will be set as the starting point.			
end	float	Optional	*None*
The timestamp below which the poses will be preserved. If set on *None*, the end of the sequence will be set as the ending point.			
use_relative_timestamps	bool	Optional	*False*
Defines if the timestamps in the output sequence will be the original ones or the timestamps relative to the first pose.			
name	str	Optional	*None*
Defines the name of the output sequence. If set on *None*, the name will be the same as the input sequence, with the suffix "+TR".			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			
add_tabs	int	Optional	0
Adds the specified amount of tabulations to the verbose outputs. This parameter can be used by other functions to encapsulate the verbose outputs by indenting them.			

Outputs

Parameter	Type
new_sequence	Sequence
A new sequence containing a subset of the poses of the original sequences.	

Sequence.trim_to_audio()

`sequence.synchronize_with_audio(delay=0, audio=None, name=None, verbose=1) -> Sequence`

Short description

Synchronizes the timestamps to the duration of an audio file.

Full description

This function returns a Sequence instance containing a subset of the poses from the original sequence. The preserved poses have a timestamp superior to *delay* seconds (if *delay* is 0, the first pose onwards will be preserved) and inferior to the duration of the *audio*. Note that this function is essentially a wrapper for the `Sequence.trim()` function, attributing *delay* and *audio*`.duration+`*delay* as the *start* and *end* parameters, respectively.

Parameters

Parameter	Type	Requirement	Default value
delay	float	Optional	*None*
The relative timestamp over which the poses will be preserved. If set on 0, the beginning of the sequence will be set as the starting point.			
audio	Audio or str	Optional	*None*
An instance of the Audio class, or a string containing the path of the audio file. In the latter case, the function will internally create an instance of the Audio class. In the case where the value is set on *None*, the function will check for a path to the audio file in the path_audio attribute (created upon initialization or via the `Sequence.set_path_audio()` method). If this attribute is also *None*, the function will raise an error.			
name	str	Optional	*None*
Defines the name of the output sequence. If set on *None*, the name will be the same as the input sequence, with the suffix "+RF".			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
new_sequence	Sequence
A new sequence containing a subset of the poses of the original sequences.	

Sequence.resample()

```
sequence.resample(frequency, mode="cubic", name=None, verbose=1) -> Sequence
```

Short description

Resamples a sequence to a fixed frequency.

Full description

This function resamples a sequence with a constant or variable framerate to the *frequency* parameter. This function also interpolates the original data to the resampled timestamps. The *mode* parameter is fed to the **scipy.interpolate.interp1d** function from the **scipy** module, that performs the resampling for the x, y and z coordinates of every joint from every pose of the sequence.

The function first creates vectors with the timestamps and the individual coordinates of the joints of the sequence, and feeds them to the **tool_functions.resample_data()** function, which returns a resampled vector and the new timestamps according to the *frequency* parameter.

Note that this function allows both to upsample and downsample sequences, and to turn sequences from variable to constant frequency. However, as it is only possible for the algorithm to *estimate* the real coordinates of the joints, you should use this function for upsampling with care. Notably, control the frequency of your original sequence with **Sequence.get_average_frequency()**, **Sequence.get_min_frequency()** and **Sequence.get_max_frequency()**.

Parameters

Parameter	Type	Requirement	Default value
frequency	float	Required	–
The frequency, in hertz, at which you want to resample the sequence. A frequency of 4 will return resample joints at 0.25 s intervals.			
mode	str	Optional	"cubic"
This parameter allows for all the values accepted for the ***kind*** parameter in the function **scipy.interpolate.interp1d**: "linear", "nearest", "nearest-up", "zero", "slinear", "quadratic", "cubic", "previous", and "next". See the documentation for this Python module for more.			
name	str	Optional	*None*
Defines the name of the output sequence. If set on *None*, the name will be the same as the input sequence, with the suffix "+RS".			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
new_sequence	Sequence
A new sequence containing joints with resampled timestamps and coordinates.	

Sequence.correct_zeros()

```
sequence.correct_zeros(mode="cubic", name=None, verbose=1, add_tabs=0) -> Sequence
```

Short description

Detects the joints set at (0, 0, 0) and correct their coordinates by interpolating the data.

Full description

This function allows to correct for the joints set at exactly (0, 0, 0) coordinates by interpolating the data from the coordinates from the surrounding temporal points. Typically, this function is used to correct the zeroes set by the Qualisys system when a specific joint is not tracked. The type of interpolation is set by the *mode* parameter. In the case where an edge pose (first or last pose of the sequence) has (0, 0, 0) coordinates, the closest non-zero coordinate is assigned to the pose. All the other joints at (0, 0, 0) coordinates are ignored for the vectors, and are inferred using the **tool_functions**.interpolate_data().

Please note that for long series of 0 coordinates, the interpolation of the data may not be accurate. When running this function with a verbosity of 1 or more, the longest duration of a 0 coordinate will be displayed.

Parameters

Parameter	Type	Requirement	Default value
mode	str	Optional	"cubic"
This parameter allows for all the values accepted for the ***kind*** parameter in the function **scipy.interpolate.interp1d**: "linear", "nearest", "nearest-up", "zero", "slinear", "quadratic", "cubic", "previous", and "next". See the documentation for this Python module for more.			
name	str	Optional	None
Defines the name of the output sequence. If set on None, the name will be the same as the input sequence, with the suffix "+CZ".			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			
add_tabs	int	Optional	0
Adds the specified amount of tabulations to the verbose outputs. This parameter can be used by other functions to encapsulate the verbose outputs by indenting them.			

Outputs

Parameter	Type
new_sequence	Sequence
A new sequence having the same amount of poses and timestamps as the original, but with re-referenced coordinates.	

Uses

- Sequence.correct_jitter()

Sequence.randomize()

```
sequence.randomize(verbose=1) -> Sequence
```

Short description

Returns a sequence that randomizes the starting position of all the joints of the original sequence.

Full description

This function creates a new sequence in which the coordinates of the joints in the first pose are randomized using a uniform distribution:

- x coordinate randomized between -0.2 and 0.2
- y coordinate randomized between -0.3 and 0.3
- z coordinate randomized between -0.5 and 0.5

The randomization preserves the direction of movement, timestamps and all of the other metrics of the sequence; only the starting position of the joints is randomized, and the coordinates of the joints of the subsequent poses are adapted using the new starting position as reference.

Parameters

Parameter	Type	Requirement	Default value
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
new_sequence	Sequence
A new sequence having the same amount of poses and timestamps as the original, but with randomized starting coordinates.	

Sequence.copy_pose()

`sequence.copy_pose(pose_index) -> Pose`

Short description

Returns a deep copy of a specified pose.

Full description

This function creates a deep copy of a pose from the sequence (i.e., creates a different Pose instance from the original) and returns it. Modifying the output pose will not modify the pose from the sequence. The index *pose_index* must be between 0 (first pose of the sequence) and n-1, n being the amount of poses in the sequence.

Parameters

Parameter	Type	Requirement	Default value
pose_index	int	Required	–
The index of the pose to be returned (between 0 and n-1, n being the amount of poses in the sequence).			

Outputs

Parameter	Type
pose	Pose
The copy of a Pose instance.	

Uses

- Sequence.trim()

Sequence.copy_joint()

`sequence.copy_joint(pose_index, joint_label) -> Joint`

Short description

Returns a deep copy of a specified joint from a specified pose.

Full description

This function creates a deep copy of a joint from a specified pose (i.e., creates a different Joint instance from the original) and returns it. Modifying the output pose will not modify the joint from the sequence. The index *pose_index* must be between 0 (first pose of the sequence) and n-1, n being the amount of poses in the sequence.

Parameters

Parameter	Type	Requirement	Default value
pose_index	int	Required	–
The index of the pose to be returned (between 0 and n-1, n being the amount of poses in the sequence).			
joint_label	str	Required	–
Label of the joint to copy (e.g. "Head").			

Outputs

Parameter	Type
joint	Joint
The copy of a Joint instance.	

Uses

- Sequence.correct_jitter()
- Sequence._correct_jitter_window()
- Sequence.re_reference()
- Sequence._create_new_sequence_with_timestamps()
- Sequence.copy_pose()

Sequence.average_qualisys_joints()

```
sequence.average_qualisys_joints(joints_labels_to_exclude=None,
remove_averaged_joints=False, remove_non_kinect_joints=False) -> None
```

Short description

Create missing Kinect joints from the Qualisys labelling system by averaging the distance between Qualisys joints.

Full description

This function creates new joints, for each pose, situated at the midpoint of the arithmetic distance between two or more joints. The list of averaged joints is set from the **`res/qualisys_to_kinect.txt`** file.

New joint	Averaged joints			
Head	HeadTop	HeadFront	HeadRight	HeadLeft
Neck	Head	SpineShoulder		
ShoulderLeft	ShoulderTopLeft	ShoulderTopRight		
ShoulderRight	ShoulderTopRight	ShoulderBackRight		
SpineMid	Chest	BackRight	BackLeft	
HipLeft	WaistBackLeft	WaistFrontLeft		
HipRight	WaistBackRight	WaistFrontRight		
SpineBase	HipLeft	HipRight		
WristLeft	WristOutLeft	WristInLeft		
WristRight	WristOutRight	WristInRight		
HandLeft	HandOutLeft	HandInLeft		
HandRight	HandOutRight	HandInRight		
FootLeft	ForefootOutLeft	ForefootInLeft	ToetipLeft	HeelLeft
FootRight	ForefootOutRight	ForefootInRight	ToetipRight	HeelRight

Parameters

Parameter	Type	Requirement	Default value
`joints_labels_to_exclude`	list	Optional	*None*
Defines the list of joint labels that will not be created from the function.			
`remove_averaged_joints`	bool	Optional	*False*
If *True*, removes the joints that are part of an averaging from every pose of the sequence.			
`remove_non_kinect_joints`	bool	Optional	*False*
If *True*, removes the joints from Qualisys that are not found in the Kinect labelling system.			

Sequence.average_joints()

```
sequence.average_joints(joints_labels_to_average, new_joint_label,
remove_averaged_joints=False) -> Joint
```

Short description

Create a joint located at the average arithmetic distance of specified joint labels.

Full description

This function creates a new joint with the label *new_joint_label*, for each pose, situated at the midpoint of the arithmetic distance between the joins specified in *joints_labels_to_average*.

New joint	Averaged joints			
Head	HeadTop	HeadFront	HeadRight	HeadLeft
Neck	Head	SpineShoulder		
ShoulderLeft	ShoulderTopLeft	ShoulderTopRight		
ShoulderRight	ShoulderTopRight	ShoulderBackRight		
SpineMid	Chest	BackRight	BackLeft	
HipLeft	WaistBackLeft	WaistFrontLeft		
HipRight	WaistBackRight	WaistFrontRight		
SpineBase	HipLeft	HipRight		
WristLeft	WristOutLeft	WristInLeft		
WristRight	WristOutRight	WristInRight		
HandLeft	HandOutLeft	HandInLeft		
HandRight	HandOutRight	HandInRight		
FootLeft	ForefootOutLeft	ForefootInLeft	ToetipLeft	HeelLeft
FootRight	ForefootOutRight	ForefootInRight	ToetipRight	HeelRight

Parameters

Parameter	Type	Requirement	Default value
joints_labels_to_exclude	list	Required	–
A list of the labels of the joints to average.			
new_joint_label	str	Required	–
The label of the newly created joint.			
remove_averaged_joints	bool	Optional	*False*
If *True*, removes the joints that are part of an averaging from every pose of the sequence.			

Sequence.concatenate()

```
sequence.concatenate(other, delay) -> None
```

Short description

Adds all the poses of another sequence at the end of the sequence.

Full description

This function adds all the poses from the sequence *other* to the poses of the current sequence. The timestamp of the first appended pose will be set on the timestamp of the original last pose of the current sequence, to which is added the parameter *delay* (in seconds). The timestamps of the subsequent poses will then be recalculated.

Parameters

Parameter	Type	Requirement	Default value
other	Sequence	Required	–
The sequence to concatenate.			
delay	float	Required	–
The delay to apply, in seconds, between the last pose of the original sequence and the first pose of the new sequence.			

Sequence.print_pose()

```
sequence.print_pose(pose_index) -> None
```

Short description

Prints the information related to one specific pose of the sequence.

Full description

This function prints the information contained in a specific pose of the sequence in a structured way. It returns the number of the pose (starting at 1), its index (starting at 0), its timestamp, and the x, y and z coordinates of each joint of the pose.

Parameters

Parameter	Type	Requirement	Default value
pose_index	int	Required	–
The index of the pose.			

Sequence.print_stats()

```
sequence.print_stats() -> None
```

Short description

Prints a series of statistics related to the sequence.

Full description

This function simply prints in a structured way and with units the output from `Sequence.get_stats()`. Refer to the documentation of this function to see the details of the outputs.

Sequence.convert_to_table()

```
sequence.convert_to_table(use_relative_timestamps=False) -> list
```

Short description

Returns a list of lists with headers, and containing the timestamps and coordinates for each joint.

Full description

This function returns a list of lists, where the first row is the headers of a table, and the subsequent rows are the values of the timestamps and coordinates of the joints of the sequence. The first column of the table contains the timestamps, while the subsequent columns, by sets of three, contain the coordinates of a joint on the x, y and z axes respectively. Here is an example of the structure of a table for a sequence with three poses and two joints:

Timestamp	Head_X	Head_Y	Head_Z	HandRight_X	HandRight_Y	HandRight_Z
0	0.400	0.80	1.50	1.60	2.300	4.2
0.125	0.1123	0.5813	0.2134	0.5589	0.1442	0.3337
0.250	0.2008	0.0512	0.1519	0.2018	0.1515	0.1300

Parameters

Parameter	Type	Requirement	Default value
use_relative_timestamps	bool	Optional	False
Defines if the timestamps in the output sequence will be the original ones or the timestamps relative to the first pose.			

Outputs

Parameter	Type
table	list
A list of lists that can be interpreted as a table, containing headers and the values of the timestamps and the coordinates of the joints from the sequence.	

Uses

- Sequence.save_xlsx()
- Sequence.save_txt()

Sequence.convert_to_json()

```
sequence.convert_to_json(use_relative_timestamps=False) -> list
```

Short description

Returns a list ready to be exported in json.

Full description

This function returns a list containing, for each element, a dictionary for each pose. The structure followed by the dictionary is the same as the output dictionary from Kinect, for compatibility reasons. Here is an example of the structure for a sequence with two poses and two joints:

```
[{“Timestamp”: 0,
“Bodies”: [
        {“Joints”:
                [
                        {“JointType”: “Head”,
                        “Position”: {“X”: 0.400, “Y”: 0.80, “Z”: 1.50}},
                        {“JointType”: “HandRight”,
                        “Position”: {“X”: 1.60, “Y”: 2.300, “Z”: 4.2}}]}]},

{“Timestamp”: 0.125,
“Bodies”: [
        {“Joints”:
                [
                        {“JointType”: “Head”,
                        “Position”: {“X”: 0.1123, “Y”: 0.5813, “Z”: 0.2134}},
                        {“JointType”: “HandRight”,
                        “Position”: {“X”: 0.5589, “Y”: 0.1442, “Z”: 0.3337}}]}]}]
```

Parameters

Parameter	Type	Requirement	Default value
use_relative_timestamps	bool	Optional	False
Defines if the timestamps in the output sequence will be the original ones or the timestamps relative to the first pose.			

Outputs

Parameter	Type
data	list
A list containing the data of the sequence, ready to be exported in json.	

Uses

- Sequence.save_json()

Sequence.save()

```
sequence.save(folder_out, name=None, file_format="json", individual=False,
use_relative_timestamp=True, verbose=1) -> Sequence
```

Short description

Saves a sequence in a file or a folder.

Full description

The function saves the sequence under *folder_out*/*name*.*file_format*. All of the non-existent subfolders present in the *folder_out* path will be created by the function.

Parameters

Parameter	Type	Requirement	Default value
folder_out	str	Required	–
The path to the folder where to save the file or files. If one or more subfolders of the path do not exist, the function will create them.			
name	str	Optional	*None*
Defines the name of the file or files where to save the sequence. If set on *None*, the name will be set on the attribute *name* of the sequence; if that attribute is also set on *None*, the name will be set on out. If *individual* is set on *True*, each pose will be saved as a different file, having the index of the pose as a suffix after the name (e.g. if the name is `"pose"` and the file format is `"txt"`, the poses will be saved as `pose_0.txt`, `pose_1.txt`, `pose_2.txt`, etc.).			
file_format	str	Optional	*None*
The file format in which to save the sequence. The file format must be `"json"`, `"xlsx"`, `"txt"`, `"csv"`, `"tsv"`, or, if you are a masochist, `"mat"`. Notes: • `"xls"` will save the file with an .xlsx extension. • Any string starting with a dot will be accepted (e.g. `".csv"` instead of `"csv"`). • `"csv;"` will force the value separator on ";", while `"csv,"` will force the separator on ",". By default, the function will detect which separator the system uses. • `"txt"` and `"tsv"` both separate the values by a tabulation. • Any other string will not return an error, but rather be used as a custom extension. The data will be saved as in a text file (using tabulations as values separators).			
individual	bool	Optional	*False*
If set on *False*, the function will save the sequence in a unique file. If set on *True*, the function will save each pose of the sequence in an individual file, appending an underscore and the index of the pose (starting at 0) after the name.			
use_relative_timestamps	bool	Optional	*False*
Defines if the timestamps in the output file(s) will be the original ones or the timestamps relative to the first pose.			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Private methods

`Sequence._define_name_init()`

Short description

Sets the name attribute for an instance of the Sequence class, using the name provided during the initialization, or the path.

Full description

Sets the name attribute for an instance of the Sequence class, by taking the provided *`name`* parameter. If no *`name`* is provided, the function will create the name based on the *`path`* provided, by defining the name as the last element of the path hierarchy (last subfolder, or file name). For example, if the *`path`* is `"C:/Users/Darlene/Documents/Recording001/"`, the function will define the name on `"Recording001"`. If both *`name`* and *`path`* are set on None, the sequence name will be defined as `"Unnamed sequence"`.

Parameters

Parameter	Type	Requirement	Default value
`name`	str	Required	–
The name passed as parameter in `Sequence.__init__()`			
`path`	str	Required	–
The path passed as parameter in `Sequence.__init__()`			
`verbose`	int	Optional	`1`
Sets how much feedback the code will provide in the console output: • `0`: **Silent mode**. The code won't provide any feedback, apart from error messages. • `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Uses

- `Sequence.__init__()`

Sequence._load_from_path()

Short description

Loads the sequence data from the path provided during the initialization, and calculates the relative timestamps from the first pose for each pose, and velocities of each joint between each pose.

Full description

Meta-function handling the loading of the data, if a path parameter has been provided in the initialisation of a Sequence instance. If the provided path is a folder, the function first calls `Sequence._fetch_files_from_folder()` to fetch the names of the valid files in the designated folder. Then, it reads the files (or single file) and create the Pose objects using the function `Sequence._load_poses()`. The function then checks if at least one pose was created; if not, it returns an error message. Finally, the function calls `Sequence._calculate_relative_timestamps()` to define timestamps starting on the first pose as reference (and, if needed, converts the timestamps to seconds) and `Sequence.calculate_velocities()` to calculate the distance over time travelled by each joint between each pose.

Parameters

Parameter	Type	Requirement	Default value
`time_unit`	str or bool	Required	–
If set on `"auto"`, the function `Sequence._calculate_relative_timestamps()` will automatically detect if to divide the timestamps by 10 000 000, 1 000, or no. If set on `"100ns"`, divides the timestamps from the file by 10 000 000. Typically, this is due to the output timestamps from Kinect being in tenth of microsecond (C# system time). If set on `"ms"`, divides the timestamps from the file by 1 000. If set on `"s"`, the function will preserve the timestamps as they are in the file. The parameter also allows other units. See the documentation for the function `Sequence._calculate_relative_timestamps()`.			
`verbose`	int	Optional	`1`
Sets how much feedback the code will provide in the console output: • `0`: **Silent mode**. The code won't provide any feedback, apart from error messages. • `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Uses

- `Sequence.__init__()`

Subfunctions called

- `Sequence._fetch_files_from_folder()`
- `Sequence._load_poses()`
- `Sequence._calculate_relative_timestamps()`
- `Sequence.calculate_velocities()`

Sequence._fetch_files_from_folder()

Short description

Finds all the files ending with the accepted extensions (.csv, .json, .tsv, .txt or .xlsx) in the folder defined by path, and orders the files according to their name.

Full description

This function first lists all the files in the folder defined by the attribute `path`. It then scans all the files, one by one:

- If an element of the directory doesn't have an extension, is a folder, or does not end with one of the accepted extensions (.csv, .json, .tsv, .txt or .xlsx), it is ignored.
- If a file has a valid extension, the function tries to detect an underscore ("_") in the name. The file names should be xxxxxx_0.ext, where xxxxxx can be any series of characters, 0 must be the index of the pose (with or without leading zeros), and ext must be an accepted extension (.csv, .json, .tsv, .txt or .xlsx). The first pose of the sequence must have the index 0. If the file does not have an underscore in the name, it is ignored. Note: the indices must be coherent with the chronological order of the timestamps.
- The function uses the number after the underscore to order the poses. This is due to differences in how file systems handle numbers without leading zeros: some place alphabetically pose_11.json before pose_2.json (1 comes before 2), while some place it after. The function converts the number after the underscore into an integer to place it properly according to its index.

The function will return an error if more than one of the accepted file types are found in the folder (e.g. .csv and .txt).

Parameters

Parameter	Type	Requirement	Default value
`verbose`	int	Optional	`1`

Sets how much feedback the code will provide in the console output:

- `0`: **Silent mode**. The code won't provide any feedback, apart from error messages.
- `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps.
- `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.

Uses

- `Sequence._load_from_path()`

Subfunctions called

None

Sequence._load_poses()

Short description

Loads all the single pose files or the global file containing all the poses.

Full description

If the provided path during the initialisation points to a folder, this function loads each file individually by calling the function `Sequence._load_single_pose_file()` for each of them. If the provided path points to a single file, calls the function `Sequence._load_sequence_file()`.

Parameters

Parameter	Type	Requirement	Default value
`verbose`	int	Optional	`1`

Sets how much feedback the code will provide in the console output:

- `0`: **Silent mode**. The code won't provide any feedback, apart from error messages.
- `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps.
- `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.

Uses

- `Sequence._load_from_path()`

Subfunctions called

- `Sequence._load_single_pose_file()`
- `Sequence._load_sequence_file()`

Sequence._load_single_pose_file()

Short description

Loads the content of a single pose file into a Pose object.

Full description

This function opens a single file passed as parameter. Depending on the file type, it will handle the content differently, but will always standardize it to generate a float called **timestamp** and a dictionary called **values**, that will then be passed to the **Sequence._create_pose()** function, which will properly create the Pose object.

- If the file extension is .json, the function will directly get the **timestamp** from data["Timestamp"] and the **values** from data["Bodies"][0]["Joints"].
- If the file extension is .xlsx, the function open_xlsx() will be called, and use the module openpyxl. The data from the Excel file is then converted using the table_to_dict_joints() function. "Timestamp" and the label names must be on the first row, and values on the second row.
- If the file extension is .csv, .tsv or .txt, the separator is defined using get_filetype_separator(), the data is loaded using open_txt(), and then converted using the table_to_dict_joints() function. "Timestamp" and the label names must be on the first row, and values on the second row.
    - In the case where the .tsv file comes from QualiSys, the algorithm checks that the file starts with "NO_OF_FRAMES". If it does, the content of the file is converted using the convert_data_from_qtm() function immediately after the call to open_txt().

Parameter	Type	Requirement	Default value
pose_number	int	Required	–
Index of the pose, defined by **Sequence._load_poses()**			
path	str	Required	–
Path of the pose file, defined by **Sequence._load_poses()**			

Uses

- **Sequence._load_poses()**

Subfunctions called

- **Sequence._create_pose()**
- **Sequence._load_date_recording()**

Sequence._load_sequence_file()

Short description

Loads the content of a global sequence file containing individual poses into Pose objects.

Full description

This function opens the file using the path provided in the **path** attribute of the Sequence instance. Depending on the file type, it will handle the content differently, but will always standardize it to generate a float called **timestamp** and a dictionary called **values** for each pose contained in the file. The function then passes each **timestamp**/**values** couple to the Sequence._create_pose() function, which will properly create the individual Pose objects.

- If the file extension is .json, the function will directly get the **timestamp** from data[X]["Timestamp"] and the **values** from data[X]["Bodies"][0]["Joints"] with X being the index of the pose.
- If the file extension is .xlsx, the function open_xlsx() will be called, and use the module openpyxl. For each pose, the data from the Excel file is then converted using the table_to_dict_joints() function. In the Excel file, "Timestamp" and the label names must be on the first row, and the values for each pose must start on the second row, with one pose per row.
- If the file extension is .csv, .tsv or .txt, the separator is defined using get_filetype_separator(), the data is loaded using open_txt(), and then converted using the table_to_dict_joints() function. In the file, "Timestamp" and the label names must be on the first row, and the values for each pose must start on the second row, with one pose per row.
    - In the case where the .tsv file comes from QualiSys, the algorithm checks that the file starts with "NO_OF_FRAMES". If it does, the content of the file is converted using the convert_data_from_qtm() function immediately after the call to open_txt().

Parameters

Parameter	Type	Requirement	Default value
verbose	int	Optional	1

Sets how much feedback the code will provide in the console output:

- 0: **Silent mode**. The code won't provide any feedback, apart from error messages.
- 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps.
- 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.

Uses

- Sequence._load_poses()

Subfunctions called

- Sequence._create_pose()
- Sequence._load_date_recording()

Sequence._create_pose()

Short description

Creates a pose based on its index, timestamp and joint information, and adds it to the Sequence instance.

Full description

This function creates a Pose object using the three parameters provided *pose_number*, *data* and *timestamp*, that have been extracted from reading a file in `Sequence._load_single_pose_file()` or `Sequence._load_sequence_file()`. Once the Pose object has been generated, it is added to the `poses` attribute.

Parameters

Parameter	Type	Requirement	Default value
`pose_number`	int	Required	–
Index of the pose, defined by `Sequence._load_poses()` or `Sequence.load_sequence_file()`.			
`data`	list	Required	–
List containing the joints dictionaries in the same order as they were in the file. Each item is a dict containing two keys: “JointType”, having for value a string matching the label of the joint, and “Position”, having for value a dict of the “X”, “Y” and “Z” coordinates (float).			

Uses

- `Sequence._load_single_pose_file()`
- `Sequence._load_sequence_file()`

Subfunctions called

None

Sequence._load_date_recording()

Short description

Loads the date of a recording from the information contained in the file(s).

Full description

This function sets the attribute `date_recording` to a datetime object if a date is saved in the file. For recordings performed with the Qualisys system, the function simply gets the date from the line starting with TIME_STAMP. For the recordings performed with Kinect, the timestamp value of the first pose is converted to a date.

Parameters

Parameter	Type	Requirement	Default value
`data`	dict or list	Required	–
The data from the file containing the recording (or the first pose)			
`verbose`	int	Optional	`1`
Sets how much feedback the code will provide in the console output: • `0`: **Silent mode**. The code won't provide any feedback, apart from error messages. • `1`: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • `2`: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Uses

- `Sequence._load_single_pose_file()`
- `Sequence._load_sequence_file()`

Subfunctions called

None

Sequence._calculate_relative_timestamps()

Short description

For all the poses of the sequence, sets and converts the `relative_timestamp` attribute taking the first pose of the sequence as reference.

Full description

This function is called internally any time a sequence is created, in order to define the timestamps of the individual poses in relation to the first pose of the sequence. It first defines the unit if the parameter *`time_unit`* is set on "auto". To do so, it checks if the difference between the timestamps of the two first poses of the sequence:

- If it is over 1000, the function presumes that the unit is in hundreds of ns (C# precision unit, Kinect output unit).
- If it is between 1 and 1000, it presumes that the unit is in milliseconds (Qualisys output unit).
- If it is below that threshold, or if there is only one pose in the sequence, it presumes that the unit is in seconds.

Note that it is possible to manually define this unit among: `"ns"`, `"1ns"`, `"10ns"`, `"100ns"`, `"µs"`, `"1µs"`, `"10µs"`, `"100µs"`, `"ms"`, `"1ms"`, `"10ms"`, `"100ms"`, `"s"`, `"sec"`, `"1s"`, `"min"`, `"mn"`, `"h"`, `"hr"`, `"d"`, `"day"`. Inputted values containing spaces and/or upper case letters are normalized.

The function then converts the timestamps to seconds and defines the relative timestamp to the first pose of the sequence for each pose, using the **`Pose._calculate_relative_timestamp()`** method. Note that the timestamps should already be in chronological order in the file or files (for the latter cases, the chronological order of the poses should follow the alphabetical naming of the files). If not,

Parameters

Parameter	Type	Requirement	Default value
`time_unit`	str	Optional	`"auto"`
If set on `"auto"`, the function automatically tries to detect if the timestamps are in hundreds of nanoseconds, in milliseconds or in seconds. Otherwise, it is possible to manually define the unit, among the following values: `"ns"`, `"1ns"`, `"10ns"`, `"100ns"`, `"µs"`, `"1µs"`, `"10µs"`, `"100µs"`, `"ms"`, `"1ms"`, `"10ms"`, `"100ms"`, `"s"`, `"sec"`, `"1s"`, `"min"`, `"mn"`, `"h"`, `"hr"`, `"d"`, `"day"`.			

Uses

- **`Sequence._load_from_path()`**
- **`Sequence.realign()`**
- **`Sequence.correct_zeros()`**
- **`Sequence.re_reference()`**
- **`Sequence.synchronize_with_audio()`**
- **`Sequence.resample()`**

Subfunctions called

- **`Pose._calculate_relative_timestamp()`**

Sequence._calculate_velocities()

Short description

Calculate the velocity compared to the previous pose, for all the joints, across all the poses.

Full description

This function calculates, for each pose and for each joint, the velocity (distance travelled over time elapsed since the previous pose, in meters per second) using the `calculate_velocity()` function, and saves it for each Pose object, using `Joint.set_velocity()`.

Uses

- `Sequence._load_from_path()`
- `Sequence.realign()`
- `Sequence.correct_zeros()`
- `Sequence.re_reference()`
- `Sequence.synchronize_with_audio()`
- `Sequence.resample()`

Subfunctions called

- `Joint._set_velocity()`

Notes

As of version 2.0, this function may disappear.

Sequence._correct_jitter_window()

Short description

Corrects linearly the jumps and twitches of the positions of a joint, between two given time points.

Full description

For a given joint *joint_label*, and two time points *start_pose_number* and *end_pose_number* between which a movement has been detected as being over threshold by the function `Sequence.correct_jitter()`, this function corrects the position of the joint between the two time points, by calling the function `Sequence._correct_jitter_single_joint()` for each individual time point. This function does not perform the correction if the size of the window is of size 0 or 1.

Parameters

Parameter	Type	Requirement	Default value
new_sequence	Sequence	Required	–
Corrected sequence that is used by and will be returned by `Sequence.correct_jitter()`.			
start_pose_number	int	Required	–
Last pose index before the jump or twitch.			
end_pose_number	int	Required	–
First pose index after the jump or twitch.			
joint_label	str	Required	–
Label of the joint to correct (e.g. "Head").			
realigned_points	int	Required	–
The number of time points corrected since the beginning of the execution of `Sequence.correct_jitter()`.			
verbose	int	Required	–
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
new_sequence	Sequence
Sequence provided in input with corrected window.	
realigned_points	int
Number of time points corrected, provided in input, incremented by the number of corrections performed.	

Uses

- `Sequence.correct_jitter()`

Subfunctions called

- `Sequence._correct_jitter_single_joint()`

Sequence._correct_jitter_single_joint()

Short description

Corrects linearly a Joint object following the algorithm in `Sequence.correct_jitter()`.

Full description

This function corrects a single joint from a jump or a twitch. Given a *joint_before* on pose *pose_before* (last pose before the jump or twitch) and *joint_after* on pose *pose_after* (first pose after the jump on twitch), this function calculates the percentage of time elapsed at *pose_current* and uses this percentage as a ratio to calculate and return the new position of the x, y and z coordinates.

Parameters

Parameter	Type	Requirement	Default value
joint_before	Joint	Required	–
Last joint before the detected jump or twitch.			
joint_after	Joint	Required	–
First joint after the detected jump or twitch.			
pose_before	int	Required	–
Index of *joint_before*.			
pose_current	int	Required	–
Index of the joint being corrected.			
pose_after	int	Required	–
Index of *joint_after*.			
verbose	int	Required	–
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Outputs

Parameter	Type
x	float
Corrected x coordinate.	
y	float
Corrected y coordinate.	
z	float
Corrected z coordinate.	

Uses

- `Sequence._correct_jitter_window()`

Sequence._create_new_sequence_with_timestamps()

Short description

Creates a sequence with the same number of poses as in the original, but only containing timestamps.

Full description

This function is noteworthily used by **Sequence**.correct_jitter() and **Sequence**.re_reference() to have placeholder poses to be modified by the function. The sequence created has the same number of Pose objects as the original sequence in the poses attribute; however, the joints attribute of these Pose is an OrderedDict that has the joint labels as keys, but *None* as values; the attributes pose_number and timestamp are preserved from the original sequence. The function finally copies the joints from the first pose only to add them to the Sequence.

Parameters

Parameter	Type	Requirement	Default value
use_relative_timestamps	bool	Optional	*False*
Last joint before the detected jump or twitch.			

Outputs

Parameter	Type
new_sequence	Sequence
A Sequence instance containing the same number of Pose objects in the pose attributes; these Pose objects have a joints attribute with all values set on *None*; the attributes pose_number and timestamp are preserved from the poses of the original sequence.	

Sequence._save_json()

```
sequence.save_json(folder_out, name=None, individual=False,
use_relative_timestamp=True, verbose=1) -> Sequence
```

Short description

Saves a sequence as a json or mat file or files.

Full description

The function saves the sequence under *folder_out*/*name*.json. All of the non-existent subfolders present in the *folder_out* path will be created by the function. The output file will contain a list containing a dictionary for each pose (if individual is set on *False*) or a dictionary of the pose (if individual is set on *True*). Here is an example of the structure for a sequence with two poses and two joints:

```
[{“Timestamp”: 0,
“Bodies”: [
        {“Joints”:
                [
                        {“JointType”: “Head”,
                        “Position”: {“X”: 0.400, “Y”: 0.80, “Z”: 1.50}},
                        {“JointType”: “HandRight”,
                        “Position”: {“X”: 1.60, “Y”: 2.300, “Z”: 4.2}}]}]},

{“Timestamp”: 0.125,
“Bodies”: [
        {“Joints”:
                [
                        {“JointType”: “Head”,
                        “Position”: {“X”: 0.1123, “Y”: 0.5813, “Z”: 0.2134}},
                        {“JointType”: “HandRight”,
                        “Position”: {“X”: 0.5589, “Y”: 0.1442, “Z”: 0.3337}}]}]}]
```

Parameters

Parameter	Type	Requirement	Default value
folder_out	str	Required	–
The path to the folder where to save the file or files. If one or more subfolders of the path do not exist, the function will create them.			
name	str	Optional	*None*
Defines the name of the file or files where to save the sequence. If set on *None*, the name will be set on the attribute *name* of the sequence; if that attribute is also set on *None*, the name will be set on out. If *individual* is set on *True*, each pose will be saved as a different file, having the index of the pose as a suffix after the name (e.g. if the name is "pose", the poses will be saved as **pose_0.json**, **pose_1.json**, **pose_2.json**, etc.).			
individual	bool	Optional	*False*
If set on *False*, the function will save the sequence in a unique file. If set on *True*, the function will save each pose of the sequence in an individual file, appending an underscore and the index of the pose (starting at 0) after the name.			
use_relative_timestamps	bool	Optional	*False*
Defines if the timestamps in the output file(s) will be the original ones or the timestamps relative to the first pose.			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won’t provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Subfunctions called

- Sequence.convert_to_json()

Sequence._save_mat()

```
sequence.save_mat(folder_out, name=None, individual=False,
use_relative_timestamp=True, verbose=1) -> Sequence
```

Short description

Saves the sequence as a Matlab .mat file or files.

Full description

The function saves the sequence under *folder_out*/*name*.mat. All of the non-existent subfolders present in the *folder_out* path will be created by the function. The output file will contain only one element, "data", which is a table. Here is an example of the structure of a table for a sequence with three poses and two joints:

Timestamp	Head_X	Head_Y	Head_Z	HandRight_X	HandRight_Y	HandRight_Z
0	0.400	0.80	1.50	1.60	2.300	4.2
0.125	0.1123	0.5813	0.2134	0.5589	0.1442	0.3337
0.250	0.2008	0.0512	0.1519	0.2018	0.1515	0.1300

Parameters

Parameter	Type	Requirement	Default value
folder_out	str	Required	–
The path to the folder where to save the file or files. If one or more subfolders of the path do not exist, the function will create them.			
name	str	Optional	*None*
Defines the name of the file or files where to save the sequence. If set on *None*, the name will be set on the attribute *name* of the sequence; if that attribute is also set on *None*, the name will be set on out. If *individual* is set on *True*, each pose will be saved as a different file, having the index of the pose as a suffix after the name (e.g. if the name is "pose", the poses will be saved as **pose_0.mat**, **pose_1.mat**, **pose_2.mat**, etc.).			
individual	bool	Optional	*False*
If set on *False*, the function will save the sequence in a unique file. If set on *True*, the function will save each pose of the sequence in an individual file, appending an underscore and the index of the pose (starting at 0) after the name.			
use_relative_timestamps	bool	Optional	*False*
Defines if the timestamps in the output file(s) will be the original ones or the timestamps relative to the first pose.			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Subfunctions called

- Sequence.convert_to_table()

Modules required

- scipy

Sequence._save_xlsx()

```
sequence.save_xlsx(folder_out, name=None, individual=False,
use_relative_timestamp=True, verbose=1) -> Sequence
```

Short description

Saves the sequence as an Excel file or Excel files.

Full description

The function saves the sequence under *folder_out*/*name*.xlsx. All of the non-existent subfolders present in the *folder_out* path will be created by the function. Here is an example of the structure of an Excel file for a sequence with three poses and two joints:

Timestamp	Head_X	Head_Y	Head_Z	HandRight_X	HandRight_Y	HandRight_Z
0	0.400	0.80	1.50	1.60	2.300	4.2
0.125	0.1123	0.5813	0.2134	0.5589	0.1442	0.3337
0.250	0.2008	0.0512	0.1519	0.2018	0.1515	0.1300

Parameters

Parameter	Type	Requirement	Default value
folder_out	str	Required	–
The path to the folder where to save the file or files. If one or more subfolders of the path do not exist, the function will create them.			
name	str	Optional	*None*
Defines the name of the file or files where to save the sequence. If set on *None*, the name will be set on the attribute *name* of the sequence; if that attribute is also set on *None*, the name will be set on out. If *individual* is set on *True*, each pose will be saved as a different file, having the index of the pose as a suffix after the name (e.g. if the name is "pose", the poses will be saved as **pose_0.xlsx**, **pose_1.xlsx**, **pose_2.xlsx**, etc.).			
individual	bool	Optional	*False*
If set on *False*, the function will save the sequence in a unique file. If set on *True*, the function will save each pose of the sequence in an individual file, appending an underscore and the index of the pose (starting at 0) after the name.			
use_relative_timestamps	bool	Optional	*False*
Defines if the timestamps in the output file(s) will be the original ones or the timestamps relative to the first pose.			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won't provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Subfunctions called

- Sequence.convert_to_table()

Modules required

- openpyxl

Sequence._save_txt()

```
sequence.save_xlsx(folder_out, name=None, file_format="csv",
individual=False, use_relative_timestamp=True, verbose=1) -> Sequence
```

Short description

Saves the sequence as a csv, txt, tsv or custom extension file or files.

Full description

The function saves the sequence under *folder_out*/*name*.*file_format*. All of the non-existent subfolders present in the *folder_out* path will be created by the function. Here is an example of the structure of a file for a sequence with three poses and two joints:

Timestamp	Head_X	Head_Y	Head_Z	HandRight_X	HandRight_Y	HandRight_Z
0	0.400	0.80	1.50	1.60	2.300	4.2
0.125	0.1123	0.5813	0.2134	0.5589	0.1442	0.3337
0.250	0.2008	0.0512	0.1519	0.2018	0.1515	0.1300

In the file, each row is separated by a new line, and each column by a value separator depending on the file format: “,” or “;” for csv, and tabulation for the other formats.

Parameters

Parameter	Type	Requirement	Default value
folder_out	str	Required	–
The path to the folder where to save the file or files. If one or more subfolders of the path do not exist, the function will create them.			
name	str	Optional	*None*
Defines the name of the file or files where to save the sequence. If set on *None*, the name will be set on the attribute *name* of the sequence; if that attribute is also set on *None*, the name will be set on out. If *individual* is set on *True*, each pose will be saved as a different file, having the index of the pose as a suffix after the name (e.g. if the name is "pose" and the file format is "csv", the poses will be saved as **pose_0.csv**, **pose_1.csv**, **pose_2.csv**, etc.).			
file_format	str	Optional	"csv"
The file format in which to save the sequence. The file format can be "txt", "csv" or "tsv". "csv;" will force the value separator on “;”, while "csv," will force the separator on “,”. By default, the function will detect which separator the system uses. "txt" and "tsv" both separate the values by a tabulation. Any other string will not return an error, but rather be used as a custom extension. The data will be saved as in a text file (using tabulations as values separators).			
individual	bool	Optional	*False*
If set on *False*, the function will save the sequence in a unique file. If set on *True*, the function will save each pose of the sequence in an individual file, appending an underscore and the index of the pose (starting at 0) after the name.			
use_relative_timestamps	bool	Optional	*False*
Defines if the timestamps in the output file(s) will be the original ones or the timestamps relative to the first pose.			
verbose	int	Optional	1
Sets how much feedback the code will provide in the console output: • 0: **Silent mode**. The code won’t provide any feedback, apart from error messages. • 1: **Normal mode**. The code will provide essential feedback such as progression markers and current steps. • 2: **Chatty mode**. The code will provide all possible information on the events happening. Note that this may clutter the output and slow down the execution.			

Subfunctions called

- Sequence.convert_to_table()

Pose

Description

Default class for poses, i.e. series of joints at a specific timestamp in time. A pose, in the motion sequence, is the equivalent of a frame in a video. The methods in this class are mainly handled by the methods in the class Sequence, but some of them can be directly accessed.

Initialisation

```
Pose(pose_number, timestamp) -> Pose
```

Short description

Creates an instance from the class Pose.

Full description

Creates an instance from the class Pose and returns a Pose object. Upon creation, only the parameter *pose_number* and *timestamp* are necessary, and are turned into attributed. The class creates an attribute joints, an empty OrderedDict that can then be filled with the method **Pose.add_joint()**.

Parameters

Parameter	Type	Requirement	Default value
pose_number	int	Required	–
The iIndex of the pose.			
timestamp	float	Required	–
The timestamp of the pose (in seconds).			

Attributes created

Attribute	Type	Value upon initialization
pose_number	int	*pose_number*
The index of the pose.		
joints	OrderedDict	OrderedDict()
A dictionary of joints, where the keys are the joint labels and the values are Joint instances.		
timestamp	float	*timestamp*
The timestamp of the pose (in seconds).		
relative_timestamp	float	*None*
The timestamp of the pose, relative to the first pose of the sequence (in seconds).		

Public methods

Pose.add_joint()

```
Pose.add_joint(joint_label, joint) -> None
```

Short description

Adds a joint to the pose.

Full description

Add a *joint* object with a label *joint_label* to the pose.

Parameters

Parameter	Type	Requirement	Default value
joint_label	str	Required	–
The label of the joint (e.g. "Head").			
joint	Joint	Required	–
An instance of the class **Joint**.			

Pose.get_joint()

```
Pose.get_joint(joint_label) -> Joint
```

Short description

Returns a joint object from the pose.

Full description

Returns the joint object having the label *joint_label* from the pose.

Parameters

Parameter	Type	Requirement	Default value
joint_label	str	Required	–
The label of the joint (e.g. "Head").			

Outputs

Parameter	Type
joint	Joint
An instance of the class **Joint**.	

Pose.generate_average_joint()

```
Pose.generate_average_joint(list_joints_to_average, new_joint_label,
add_joint=True) -> Joint
```

Short description

Generates and returns a joint that is located at the average position of the other joints.

Full description

This function creates a new joint with the label *new_joint_label*, of which the x, y, and z coordinates are the average x, y and z coordinates respectively of all the joints in the list of labels *list_joints_to_average*. If *add_joint* is set on *True*, the joint created is also added to the parameter joints of the current Pose instance.

Parameters

Parameter	Type	Requirement	Default value
list_joints_to_average	list	Required	–
A list containing the strings of the joints to average.			
new_joint_label	str	Required	–
The label of the joint (e.g. "Head").			
add_joint	bool	Optional	*True*
If set on *True*, the joint created is also added to the parameter joints of the current Pose instance.			

Outputs

Parameter	Type
joint	Joint
The averaged joint.	

Pose.remove_joint()

```
Pose.remove_joint(joint_label) -> None
```

Short description

Removes the specified joint from the pose.

Full description

This function removes the joint specified by *joint_label* from the parameter joints of the current Pose instance.

Parameters

Parameter	Type	Requirement	Default value
joint_label	str	Required	–
The label of the joint to remove (e.g. “Head”).			

Pose.remove_joints()

```
Pose.remove_joints(joint_labels) -> None
```

Short description

Removes the specified joints from the pose.

Full description

This function removes the joints specified by *joint_labels* from the parameter joints of the current Pose instance.

Parameters

Parameter	Type	Requirement	Default value
joint_labels	list	Required	–
A list of labels of joints to remove.			

Pose.set_timestamp()

```
Pose.set_timestamp(timestamp) -> None
```

Short description

Sets the timestamp of the pose (in seconds).

Full description

Sets the timestamp of the pose (in seconds).

Parameters

Parameter	Type	Requirement	Default value
timestamp	float	Required	–
The timestamp to assign to the pose (in seconds).			

Pose.get_timestamp()

```
Pose.get_timestamp() -> float
```

Short description

Returns the timestamp of the pose (in seconds).

Full description

Returns the timestamp of the pose (in seconds).

Outputs

Parameter	Type
timestamp	float
The timestamp of the pose, in seconds.	

Pose.get_relative_timestamp()

```
Pose.get_relative_timestamp() -> float
```

Short description

Returns the timestamp of the pose, relative to the first pose of the sequence (in seconds).

Full description

Returns the timestamp of the pose, relative to the first pose of the sequence (in seconds).

Outputs

Parameter	Type
relative_timestamp	float
The timestamp of the pose relative to the first timestamp of the sequence, in seconds.	

Pose.convert_to_table()

```
Pose.convert_to_table(use_relative_timestamps=False) -> list
```

Short description

Full description

Outputs

Parameter	Type
table	list

Private methods

Pose._calculate_relative_timestamp()

```
Pose._calculate_relative_timestamp(t, time_unit="s") -> Joint
```

Short description

Calculates the timestamp relative to the first pose of the sequence.

Full description

This function, typically called at the end of the initialisation of a new sequence (either by opening a file or performing a processing on an existing sequence), automatically calculates the timestamp relative to the first pose of the sequence, and assigns it to the attribute `relative_timestamp`.

Parameters

Parameter	Type	Requirement	Default value
`t`	float	Required	–
The timestamp of the first pose of the sequence.			
`time_unit`	str	Optional	`"s"`
The time unit of the timestamps of the sequence. This parameter can take the following values: among the following values: `"ns"`, `"1ns"`, `"10ns"`, `"100ns"`, `"µs"`, `"1µs"`, `"10µs"`, `"100µs"`, `"ms"`, `"1ms"`, `"10ms"`, `"100ms"`, `"s"`, `"sec"`, `"1s"`, `"min"`, `"mn"`, `"h"`, `"hr"`, `"d"`, `"day"`.			

Joint
Full descriptions of the methods and attributes of this class will be added in a future update.

Graphic Display
Full descriptions of the methods and attributes of this class will be added in a future update.

Graphic Sequence
Full descriptions of the methods and attributes of this class will be added in a future update.

Graphic Pose
Full descriptions of the methods and attributes of this class will be added in a future update.

Graphic Joint
Full descriptions of the methods and attributes of this class will be added in a future update.

Graphic Line
Full descriptions of the methods and attributes of this class will be added in a future update.

Time
Full descriptions of the methods and attributes of this class will be added in a future update.

Graph
Full descriptions of the methods and attributes of this class will be added in a future update.

GraphPlot
Full descriptions of the methods and attributes of this class will be added in a future update.

Audio
Full descriptions of the methods and attributes of this class will be added in a future update.

Functions

Core functions

Full descriptions of these 25 functions will be added in a future update.

Graphic functions

Full descriptions of these 6 functions will be added in a future update.

Plot functions

Full descriptions of these 5 functions will be added in a future update.

Stats functions

Full descriptions of these 16 functions will be added in a future update.

Tool functions

Full descriptions of these 34 functions will be added in a future update.

To do

- Add audio support to sequence_reader functions. Link this to the path_audio defined in Sequence.
- Make realigner with a non-linear option
- Make realigner with a time option – with closest value
- Change realigner name to “correct_jitter”
- Add get_poses() method
- Add person’s height in data stats
- Add get_joint_as_list method
- Documentation for the functions of Sequence
- Add distance between two hands as metric
- Add audio to video generation
- Add correction for Qualisys videos
- Add pitch
- Add formants
- Check that the difference is made between joint and joint_label (checked in Sequence)
- Add output tables
- Visualisation: add angle of view (front, back, left, right, top, bottom)
- Check uses and subfunctions
- Copy functions short descriptions in the Python file (checked in Sequence)
- Check use of relative timestamps and normal timestamps.
- Simplify the trim function
- Add function get_pose
- Add date of recording extraction
- Reintroduce saving with original sequence?
- Add .mat as saving extension
- Test mat saving
- Add .mat as opening extension
- Add dependencies
- Check that the modules imports are handled properly (checked in Sequence)

Ideas

- Function to automatically order sequences by date of recording