Package: PhysioMoCap (via r-universe)

May 16, 2026

Title Motion Capture and Biomechanics Analysis for PhysioExperiment Objects

Version 0.2.0

Description Provides motion capture, pose estimation, and biomechanical simulation analysis for PhysioExperiment objects. Includes I/O for C3D, Venus3D, GaitRec, OpenPose, DeepLabCut, MediaPipe, BVH, ASF/AMC, OpenSim, and generic CSV formats. Provides marker tracking (Hungarian algorithm), signal filtering, numerical differentiation, resampling, center of mass calculation, gait parameters, clinical statistics (ICC, SEM, MDC, Bland-Altman), movement phase segmentation, dynamic time warping, movement normalization, functional PCA, force-plate kinetics (GRF filtering, COP, loading rate, impulse), planar inverse dynamics (joint moments/power), EMG integration (rectification, RMS envelope, MVC normalization), gait/running/jump task schemas, OpenSim workflow integration, and biomechanics-specific visualization.

Depends R (>= 4.2), PhysioCore

Imports SummarizedExperiment, S4Vectors, stats, utils, jsonlite, ggplot2, rlang, scales

Suggests PhysioIO, signal, signalIO, uwot, Rtsne, c3dr, clue, rhdf5, httr, withr, knitr, rmarkdown, testthat (>= 3.2.0)

VignetteBuilder knitr

License MIT + file LICENSE

URL https://github.com/matsui-lab/PhysioExperiment

BugReports https://github.com/matsui-lab/PhysioExperiment/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Collate 'class-TaskSchema.R' 'schemas-prebuilt.R' 'mocap-skeleton.R' 'mocap-angles.R' 'mocap-gaps.R' 'mocap-imu.R' 'mocap-tracking.R' 'io-openpose.R' 'io-opensim.R' 'io-c3d.R' 'io-bvh.R' 'io-deeplabcut.R' 'io-mediapipe.R' 'io-opencap.R'

'ops-events.R' 'ops-phases.R' 'ops-normalize.R' 'ops-dtw.R'
'ops-dimred.R' 'stats-fda.R' 'opensim-integration.R'
'vis-biomech.R' 'vis-movement.R' 'vis-skeleton.R'
'biomech-com.R' 'io-asfamc.R' 'io-csv.R' 'io-gaitrec.R'
'io-venus3d.R' 'ops-filter.R' 'ops-derivatives.R'
'ops-resample.R' 'stats-clinical.R' 'stats-gait.R'
'kinetics-forceplate.R' 'kinetics-inverse-dynamics.R'
'mocap-emg.R' 'user-onboarding.R' 'benchmark-validation.R'
'zzz.R'

RoxygenNote 7.3.3

Config/pak/sysreqs zlib1g-dev

Repository https://x-biosignal.r-universe.dev

Date/Publication 2026-03-16 11:31:37 UTC

RemoteUrl https://github.com/x-biosignal/PhysioMoCap

RemoteRef HEAD

RemoteSha 7047593a8caad70fa3804c733be7b31b759fea92

Contents

alignEMGtoMoCap . . . 6
analyzeForcePlate . . . 7
analyzeForcePlatePE . . . 8
assessMoCapReadiness . . . 9
batch_analyze_opensim . . . 10
batchNormalize . . . 12
benchmarkAgreement . . . 13
benchmarkManifestTemplate . . . 14
blandAltman . . . 14
butterworthFilter . . . 16
calculateCOM . . . 17
calculateCOP . . . 18
calculateGaitParameters . . . 19
calculateJointAngles . . . 21
calculateSegmentCOM . . . 22
calculateStepSymmetry . . . 23
calibrateIMU . . . 24
cohensD . . . 26
combineTrials . . . 27
computeAcceleration . . . 28
computeImpulse . . . 29
computeJerk . . . 30
computeJointPower . . . 31
computeLoadingRate . . . 31
computeRMSEnvelope . . . 32
computeSpeed . . . 33
computeVelocity . . . 34

correctSwaps . . . 35
create_schema_from_opensim . . . 36
createBenchmarkExample . . . 37
defaultBenchmarkThresholds . . . 39
define_skeleton . . . 39
demoMoCapData . . . 40
detectEvents . . . 42
detectForcePlateContacts . . . 43
detectGaps . . . 44
detectSwaps . . . 45
differentiate . . . 46
dtwAverage . . . 47
dtwClustering . . . 48
dtwDistance . . . 49
dtwDistanceMatrix . . . 51
dtwWarp . . . 52
estimateOrientation . . . 53
estimateSegmentInertia . . . 54
etaSquared . . . 55
eulerToQuaternion . . . 56
Event . . . 57
extractPhase . . . 59
extractWaveformFeatures . . . 60
fillGaps . . . 61
fillGapsLinear . . . 62
fillGapsSpline . . . 63
filterGRF . . . 64
filterSignals . . . 65
fPCA . . . 66
get_bone_connections . . . 67
get_limb_pairs . . . 68
get_segment_lengths . . . 69
getEvent . . . 70
getEventNames . . . 71
getPhase . . . 71
getPhaseColors . . . 72
getPhaseData . . . 73
getPhaseNames . . . 73
getSchema . . . 74
hasValidPhases . . . 75
icc . . . 75
integrateEMGMoCap . . . 77
inverseDynamics2D . . . 78
inverseDynamics3D . . . 79
listSchemas . . . 81
manualEvents . . . 82
mdc . . . 83
movingAverage . . . 84

normalizedTimeAxis . . . 85
normalizeEMG . . . 86
normalizeMovement . . . 87
Phase . . . 88
phaseDurations . . . 89
phaseRatios . . . 90
phaseTiming . . . 90
plotCorrelationMatrix . . . 91
plotCycle . . . 92
plotDTW . . . 94
plotEffectSizeForest . . . 95
plotFPCA . . . 96
plotGaitCycle . . . 97
plotGroupComparison . . . 98
plotMultiPanel . . . 99
plotPCAScatter . . . 100
plotPCAVariance . . . 101
plotPhaseDurations . . . 101
plotPhasePortrait . . . 102
plotSkeleton . . . 103
plotSkeleton3D . . . 105
plotSkeletonOverlay . . . 106
plotSkeletonSequence . . . 108
plotSpaghetti . . . 109
plotSymmetry . . . 110
plotTrajectory . . . 111
plotUMAP . . . 112
plotWaveformComparison . . . 113
print.ASFSkeleton . . . 114
print.benchmark_agreement . . . 115
print.benchmark_manifest_validation . . . 116
print.benchmark_suite . . . 116
print.detected_events . . . 117
print.dtw_clustering . . . 117
print.dtw_result . . . 118
print.forceplate_analysis . . . 118
print.forceplate_analysis_multi . . . 119
print.fpca_result . . . 119
print.gait_parameters . . . 120
print.mocap_quickstart . . . 121
print.mocap_readiness . . . 121
print.multi_trial_phases . . . 122
print.segmented_phases . . . 122
print.SkeletonModel . . . 123
print.waveform_pca . . . 123
print.waveform_umap . . . 124
processEMG . . . 124
projectTo2D . . . 125

quaternionToEuler . . . 126
quickStartMoCap . . . 127
readAMC . . . 129
readASF . . . 130
readBVH . . . 131
readC3D . . . 132
readDeepLabCut . . . 133
readMediaPipe . . . 134
readMoCapAuto . . . 136
readMoCapCSV . . . 137
readMOT . . . 139
readOpenCap . . . 140
readOpenPose . . . 141
readOpenSimOutputs . . . 143
readSTO . . . 143
readTRC . . . 144
readVenus3D . . . 145
reconstructFPCA . . . 146
rectifyEMG . . . 147
registerCurves . . . 148
removeGravity . . . 149
reportGaps . . . 150
resampleSignal . . . 151
resampleVector . . . 152
run_opensim_toolchain . . . 153
runBenchmarkSuite . . . 154
savgolFilter . . . 155
schema_balance . . . 156
schema_cutting . . . 157
schema_cycling . . . 157
schema_gait . . . 158
schema_jump . . . 159
schema_running . . . 159
schema_throw . . . 160
segmentParameters . . . 161
segmentPhases . . . 162
sem . . . 163
SkeletonModel . . . 164
summarizeGaitParameters . . . 165
symmetryIndex . . . 166
synchronizeSignals . . . 167
TaskSchema . . . 168
trackMarkers . . . 170
validateBenchmarkManifest . . . 171
validateSchema . . . 172
vectorAngle . . . 173
waveformPCA . . . 174
waveformTSNE . . . 175

waveformUMAP 176
writeBenchmarkManifest 177

Index **179**

`alignEMGtoMoCap` *Align EMG to motion-capture sampling grid*

Description

Align EMG to motion-capture sampling grid

Usage

```
alignEMGtoMoCap(
  emg,
  emg_sampling_rate,
  mocap_length,
  mocap_sampling_rate,
  method = "linear"
)
```

Arguments

`emg`	Numeric vector or matrix of EMG data (time x channels).
`emg_sampling_rate`	EMG sampling rate (Hz).
`mocap_length`	Target number of MoCap samples.
`mocap_sampling_rate`	Target MoCap sampling rate (Hz).
`method`	Interpolation method passed to `stats::approx()`.

Value

Matrix of aligned EMG data with `mocap_length` rows.

References

Merletti R, Parker PA (2004). "Electromyography: Physiology, Engineering, and Non-Invasive Applications." IEEE Press/Wiley.

See Also

`integrateEMGMoCap()` for full EMG-MoCap integration, `resampleSignal()` for general signal resampling.

Examples

```
emg <- matrix(rnorm(2000), ncol = 2)
emg_aligned <- alignEMGtoMoCap(emg, 1000, mocap_length = 200,
                               mocap_sampling_rate = 100)
```

`analyzeForcePlate` *Run complete force-plate analysis*

Description

Convenience wrapper that filters GRF components and computes COP, loading rate, and impulse summaries.

Usage

```
analyzeForcePlate(
  forces,
  moments = NULL,
  sampling_rate,
  cutoff = 20,
  threshold = 20,
  order = 4,
  filter_method = c("butterworth", "moving_average"),
  origin = c(0, 0, 0)
)
```

Arguments

`forces`	Matrix/data.frame with force components (X, Y, Z).
`moments`	Optional matrix/data.frame with moment components (X, Y, Z).
`sampling_rate`	Sampling rate in Hz.
`cutoff`	Low-pass cutoff for force filtering.
`threshold`	Contact threshold for stance detection.
`order`	Butterworth filter order.
`filter_method`	Filtering method passed to `filterGRF()`.
`origin`	Force-plate origin used in COP computation.

Value

A list with elements: `filtered_forces`, `cop`, `loading_rate`, `impulse`, and `summary`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`filterGRF()` for ground reaction force filtering, `calculateCOP()` for center of pressure computation, `computeLoadingRate()` for loading rate analysis, `analyzeForcePlatePE()` for PhysioExperiment-based analysis.

Examples

```
f <- cbind(Fx = rep(0, 1000), Fy = rep(0, 1000),
           Fz = c(rep(0, 200), abs(sin(seq(0, pi, length.out = 600))) * 800,
                  rep(0, 200)))
out <- analyzeForcePlate(f, sampling_rate = 1000)
```

---

analyzeForcePlatePE *Analyze force-plate signals from a PhysioExperiment object*

Description

Convenience wrapper for `analyzeForcePlate()` when force and moment components are stored in PhysioExperiment assays.

Usage

```
analyzeForcePlatePE(
  pe,
  force_assay = "grf",
  moment_assay = NULL,
  plate_index = 1L,
  sampling_rate = NULL,
  ...
)
```

Arguments

`pe`	A PhysioExperiment object.
`force_assay`	Assay name containing force data with 3 columns (X, Y, Z). If not present, the function tries `force_x`, `force_y`, and `force_z` assays.
`moment_assay`	Optional assay name containing moments with 3 columns. If not present, the function tries `moment_x`, `moment_y`, and `moment_z`.
`plate_index`	Which plate to analyze when components are provided as separate assays with multiple force plates. Accepts: • numeric scalar (specific plate), • `"auto"` (select plate with largest vertical-force impulse), • `"all"` (analyze all plates and return a multi-plate result).
`sampling_rate`	Optional sampling rate in Hz. If `NULL`, uses `samplingRate(pe)`.
`...`	Additional arguments passed to `analyzeForcePlate()`.

Value

A `forceplate_analysis` object (single plate) or `forceplate_analysis_multi` (when `plate_index = "all"`).

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`analyzeForcePlate()` for matrix-based force plate analysis, `print.forceplate_analysis()` for displaying results.

Examples

```
## Not run:
pe <- readC3D("trial.c3d", include_analog = TRUE)
out <- analyzeForcePlatePE(pe)

## End(Not run)
```

---

assessMoCapReadiness *Assess readiness of a MoCap dataset for downstream analysis*

Description

Performs a compact quality and metadata checklist so beginners can quickly identify missing requirements before running kinematics or kinetics.

Usage

```
assessMoCapReadiness(
  pe,
  required_assays = c("position_x", "position_y", "position_z"),
  min_frames = 100,
  min_markers = 5,
  min_sampling_rate = 50,
  max_missing_rate = 0.05
)
```

Arguments

`pe`	A `PhysioExperiment` object.
`required_assays`	Character vector of required assay names.
`min_frames`	Minimum recommended number of frames.

`min_markers` Minimum recommended number of markers/channels.

`min_sampling_rate`
Minimum recommended sampling rate (Hz).

`max_missing_rate`
Maximum allowed missing-value rate per required assay.

Value

An S3 object of class `"mocap_readiness"` with score, grade, checks, and summary metrics.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`print.mocap_readiness()` for formatted display of readiness results, `quickStartMoCap()` for complete getting-started workflow, `demoMoCapData()` for generating demo data.

Examples

```
demo <- demoMoCapData(seed = 1)
report <- assessMoCapReadiness(demo$mocap)
report
```

---

batch_analyze_opensim *Batch analyze OpenSim session data*

Description

Loads an OpenSim session and performs batch movement analysis using a TaskSchema for event detection and normalization.

Usage

```
batch_analyze_opensim(
  session_dir,
  schema,
  signal_mapping = list(),
  trials = NULL,
  normalize_method = NULL,
  ...
)
```

Arguments

session_dir	Directory containing OpenSim output files, or an opensim_session/opensim_workflow object from signalIO
schema	A TaskSchema object defining events and phases
signal_mapping	Named list mapping schema signal names to actual channel names in the data
trials	Character vector of trial names to analyze. If NULL, analyzes all.
normalize_method	Override schema normalization method
...	Additional arguments passed to normalizeMovement

Value

A list containing:

- trials: List of normalized trial data
- events: List of detected events per trial
- phases: List of phase timing per trial
- summary: Summary statistics across trials

References

Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, Guendelman E, Thelen DG (2007). "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement." IEEE Transactions on Biomedical Engineering, 54(11), 1940-1950.

See Also

create_schema_from_opensim() for creating task schemas from OpenSim models.

Examples

```
## Not run:
library(signalIO)

# Load session and analyze
results <- batch_analyze_opensim(
  "subject01/",
  schema = schema_gait,
  signal_mapping = list(vGRF = "ground_force_vy_r")
)

# Access normalized trials
trial1 <- results$trials[[1]]

# View summary
print(results$summary)

## End(Not run)
```

`batchNormalize` *Batch normalize multiple trials*

Description

Normalizes multiple trials and returns them in a consistent format.

Usage

```
batchNormalize(
  trials,
  method = "cycle",
  norm_length = 101L,
  schema = NULL,
  ...
)
```

Arguments

`trials`	List of matrices or PhysioExperiment objects
`method`	Normalization method
`norm_length`	Target length
`schema`	Optional TaskSchema
`...`	Additional arguments

Value

3D array (time x channels x trials) or list

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`normalizeMovement()`, `normalizedTimeAxis()`, `combineTrials()`

benchmarkAgreement *Compute agreement metrics against a reference signal table*

Description

Computes per-variable metrics (RMSE, MAE, bias, correlation, R2, ICC, Bland-Altman LoA width) between prediction and reference data.

Usage

```
benchmarkAgreement(
  prediction,
  reference,
  trial_id = "trial_1",
  thresholds = defaultBenchmarkThresholds("balanced"),
  alignment = c("truncate", "resample")
)
```

Arguments

prediction	Numeric vector/matrix/data.frame.
reference	Numeric vector/matrix/data.frame.
trial_id	Label used in output rows.
thresholds	Optional named list from defaultBenchmarkThresholds().
alignment	Alignment mode when sample lengths differ: "truncate" or "resample".

Value

Object of class "benchmark_agreement" with metrics, summary, and thresholds.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

Bland JM, Altman DG (1986). "Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement." Lancet, 327(8476), 307-310.

See Also

defaultBenchmarkThresholds() for threshold configuration, runBenchmarkSuite() for multi-trial benchmarking, blandAltman() for Bland-Altman agreement analysis.

Examples

```
ref <- data.frame(a = sin(seq(0, 1, length.out = 100)))
pred <- ref + rnorm(100, sd = 0.01)
out <- benchmarkAgreement(pred, ref, trial_id = "demo")
out$summary
```

benchmarkManifestTemplate *Create a benchmark manifest template*

Description

Produces a template data.frame describing benchmark trials and file paths.

Usage

```
benchmarkManifestTemplate(n = 1L)
```

Arguments

n	Number of template rows to create.

Value

A data.frame manifest template.

References

Bland JM, Altman DG (1986). "Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement." Lancet, 327(8476), 307-310.

See Also

`writeBenchmarkManifest()` for writing the template to CSV, `validateBenchmarkManifest()` for validating manifest structure.

Examples

```
benchmarkManifestTemplate(2)
```

blandAltman *Bland-Altman Analysis for Method Agreement*

Description

Performs a Bland-Altman analysis comparing two measurement methods or two time points. Computes the bias (mean difference), limits of agreement, and confidence interval for the bias.

Usage

```
blandAltman(x, y, confidence = 0.95)
```

Arguments

x	Numeric vector of measurements from method/time 1.
y	Numeric vector of measurements from method/time 2.
confidence	Numeric; confidence level for limits of agreement (default 0.95).

Details

The Bland-Altman method assesses agreement between two measurements by plotting their difference against their mean. The limits of agreement are:

$$LoA = \bar{d} \pm z \times SD_d$$

where $\bar{d}$ is the mean difference (bias) and SD_d is the SD of differences.

Value

A list with components:

bias Mean difference (x - y)

lower_loa Lower limit of agreement

upper_loa Upper limit of agreement

sd_diff Standard deviation of differences

ci_bias Two-element vector with lower and upper CI for the bias

References

Bland JM, Altman DG (1986). Statistical methods for assessing agreement between two methods of clinical measurement. Lancet, 327(8476), 307-310.

See Also

`icc()` for intraclass correlation reliability analysis, `benchmarkAgreement()` for benchmark validation agreement metrics.

Examples

```
set.seed(42)
method1 <- rnorm(30, mean = 50, sd = 10)
method2 <- method1 + rnorm(30, mean = 0, sd = 3)
result <- blandAltman(method1, method2)
result$bias
result$lower_loa
result$upper_loa
```

`butterworthFilter` *Apply a Butterworth filter to numeric data*

Description

Designs a Butterworth IIR filter and applies it using zero-phase filtering (forward-backward) to avoid phase distortion.

Usage

```
butterworthFilter(
  x,
  cutoff,
  sampling_rate,
  type = c("lowpass", "highpass", "bandpass", "bandstop"),
  order = 4
)
```

Arguments

`x`	A numeric vector or matrix (time x channels).
`cutoff`	Cutoff frequency in Hz. A single value for "lowpass" or "highpass"; a length-2 vector `c(low, high)` for "bandpass" or "bandstop".
`sampling_rate`	Sampling rate in Hz.
`type`	Filter type: "lowpass", "highpass", "bandpass", or "bandstop".
`order`	Filter order (default: 4).

Details

Requires the **signal** package. NAs in the input are temporarily interpolated before filtering and restored afterward.

Value

Filtered data with the same dimensions as `x`.

References

Butterworth S (1930). "On the Theory of Filter Amplifiers." Wireless Engineer, 7, 536-541.

See Also

`filterSignals()` for filtering PhysioExperiment objects, `savgolFilter()` for Savitzky-Golay polynomial smoothing.

Examples

```
## Not run:
# Filter a single vector
x <- sin(2 * pi * 5 * seq(0, 1, length.out = 1000)) +
     sin(2 * pi * 50 * seq(0, 1, length.out = 1000))
x_filt <- butterworthFilter(x, cutoff = 20, sampling_rate = 1000, type = "lowpass")

## End(Not run)
```

`calculateCOM` *Calculate whole-body center of mass from marker positions*

Description

Computes the whole-body center of mass (COM) at each time frame from 3D (or 2D) marker position data and body segment inertial parameters.

Usage

```
calculateCOM(pe, body_mass, skeleton = NULL, bsip = NULL, marker_map = NULL)
```

Arguments

`pe`	A `PhysioExperiment` object containing `position_x`, `position_y`, and optionally `position_z` assays.
`body_mass`	Numeric scalar, body mass in kilograms. Must be positive.
`skeleton`	Optional `SkeletonModel` object used for automatic marker mapping. If provided and `marker_map` is `NULL`, a default mapping is generated from the skeleton model.
`bsip`	Optional data.frame of body segment parameters (as returned by `segmentParameters()`). If `NULL` (default), uses `segmentParameters("deLeva_male")`.
`marker_map`	Optional named list mapping each segment to its proximal and distal marker names. Each element should be a character vector of length 2: `c(proximal_marker, distal_marker)`. Segment names must match those in the `bsip` table. Example: `list(thigh_r = c("RHip", "RKnee"), thigh_l = c("LHip", "LKnee"))`.

Details

The algorithm proceeds as follows:

1. For each body segment, compute the segment COM position as:

$$COM_{seg} = P_{prox} + f \times (P_{dist} - P_{prox})$$

   where f is the COM proximal fraction from the BSIP table.

2. Compute the whole-body COM as the mass-weighted average of all segment COMs:

$$COM_{body} = \frac{\sum_i m_i \times COM_i}{M}$$

   where $m_i = M \times f_i$ is the segment mass and M is the total body mass.

Value

A PhysioExperiment object with additional assays com_x, com_y, and (if 3D) com_z. Each assay is a single-column matrix with column name "COM" and the same number of rows as the input.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

Zatsiorsky VM (2002). "Kinetics of Human Motion." Human Kinetics.

See Also

segmentParameters() for body segment inertial parameters, calculateSegmentCOM() for individual segment center of mass, symmetryIndex() for bilateral symmetry assessment.

Examples

```
## Not run:
pe <- readTRC("markers.trc")
result <- calculateCOM(pe, body_mass = 75)
com_x <- SummarizedExperiment::assay(result, "com_x")

## End(Not run)
```

---

calculateCOP *Calculate center of pressure (COP) from force-plate data*

Description

Computes COP coordinates from force and moment components using standard force-plate equations (assuming a vertical Z axis).

Usage

```
calculateCOP(forces, moments, origin = c(0, 0, 0), min_vertical_force = 20)
```

Arguments

forces	Matrix/data.frame with at least three columns for force components (X, Y, Z).
moments	Matrix/data.frame with at least three columns for moment components (X, Y, Z).
origin	Numeric length-3 vector with force-plate origin (x, y, z).
min_vertical_force	Minimum absolute vertical force required to compute COP (samples below this threshold are set to NA).

Value

A data.frame with columns `cop_x`, `cop_y`, `cop_z`, `free_moment`, and `vertical_force`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`analyzeForcePlate()` for comprehensive force plate analysis, `calculateCOM()` for whole-body center of mass computation.

Examples

```
f <- cbind(Fx = rep(0, 10), Fy = rep(0, 10), Fz = rep(1000, 10))
m <- cbind(Mx = rep(100, 10), My = rep(-200, 10), Mz = rep(0, 10))
cop <- calculateCOP(f, m)
```

---

calculateGaitParameters

Calculate gait temporal-spatial parameters

Description

Computes temporal and spatial gait parameters from detected gait events. Temporal parameters include stride time, step time, stance/swing durations, double support time, and cadence. Spatial parameters (step length, stride length, step width, walking speed) require position data in the PhysioExperiment assays.

Usage

```
calculateGaitParameters(
  pe,
  events,
  body_height = NULL,
  side = c("right", "left", "both")
)
```

Arguments

`pe`	A PhysioExperiment object with position data (assays named `position_x`, `position_y`, or `position_z`).
`events`	A `detected_events` object (from `detectEvents()`) or a data.frame with at minimum columns `event_name` (or `event`) and `time` (sample indices or seconds), plus `side` (`"right"` or `"left"`).

body_height	Numeric; body height in metres for normalized parameters (optional). Currently reserved for future use.
side	Character; which side(s) to compute. One of "right", "left", or "both" (default).

Details

Events must use one of the recognised naming conventions:

- "HS_R", "HS_L" / "TO_R", "TO_L"
- "right_heel_strike", "left_heel_strike" / "right_toe_off", "left_toe_off"

The events data.frame may use either event_name or event for the event name column, and either time (in sample indices) or time_sec (in seconds) for timing.

Value

An S3 object of class "gait_parameters" inheriting from data.frame. Each row is one stride cycle. Columns include:

side Side label ("right" or "left")

stride Stride number (sequential within side)

stride_time Time between consecutive ipsilateral heel strikes (s)

step_time Time from contralateral to ipsilateral heel strike (s)

stance_time Heel strike to toe off on the same side (s)

swing_time Toe off to next heel strike on the same side (s)

stance_percent Stance as percentage of stride time

swing_percent Swing as percentage of stride time

double_support_time Time with both feet on the ground (s)

cadence Steps per minute (60 / step_time)

step_length AP distance between feet at consecutive heel strikes (m)

stride_length AP distance between ipsilateral heel strikes (m)

step_width ML distance between feet at heel strike (m)

walking_speed Stride length / stride time (m/s)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

Perry J, Burnfield JM (2010). "Gait Analysis: Normal and Pathological Function." 2nd ed. SLACK Incorporated.

See Also

calculateStepSymmetry() for symmetry analysis of gait parameters, summarizeGaitParameters() for descriptive statistics of gait data, plotGaitCycle() for gait cycle visualization.

Examples

```
# See test file for worked examples with synthetic data
```

`calculateJointAngles` *Calculate joint angles from marker positions*

Description

Computes joint angles from 3D (or 2D) position data stored in a PhysioExperiment object. Each joint is defined by three points: proximal, joint (vertex), and distal. The angle at the vertex is computed using the specified convention.

Usage

```
calculateJointAngles(pe, joints, convention = c("3point"), degrees = TRUE)
```

Arguments

`pe`	A `PhysioExperiment` object with position assays (`position_x`, `position_y`, and optionally `position_z`).
`joints`	A named list of joint definitions. Each element is a list with components `proximal`, `joint`, and `distal`, which can be column names (character) or column indices (integer) referring to the position assays.
`convention`	Character string specifying the angle convention. Currently only `"3point"` is supported. Default `"3point"`.
`degrees`	Logical. If `TRUE` (default), return angles in degrees. If `FALSE`, return angles in radians.

Details

For the `"3point"` convention, the angle is computed at the vertex (joint point) between the proximal-to-joint and distal-to-joint direction vectors:

$$\theta = \arccos\left(\frac{v_1 \cdot v_2}{|v_1||v_2|}\right)$$

where $v_1 = \text{proximal} - \text{joint}$ and $v_2 = \text{distal} - \text{joint}$.

The result is clamped to $[0, \pi]$ (or $[0, 180]$ in degrees). If any of the three points contain `NA` for a given frame, the angle for that frame is `NA`.

Value

A `PhysioExperiment` object with an additional `"joint_angles"` assay containing the computed angles (n_frames x n_joints). The column names of the assay correspond to the names of the `joints` list. The `colData` is updated with joint angle metadata.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

Grood ES, Suntay WJ (1983). "A joint coordinate system for the clinical description of three-dimensional motions: application to the knee." Journal of Biomechanical Engineering, 105(2), 136-144.

See Also

vectorAngle(), quaternionToEuler(), eulerToQuaternion()

Examples

```
## Not run:
pe <- readTRC("markers.trc")
joints <- list(
  right_elbow = list(
    proximal = "RShoulder",
    joint    = "RElbow",
    distal   = "RWrist"
  ),
  right_knee = list(
    proximal = "RHip",
    joint    = "RKnee",
    distal   = "RAnkle"
  )
)
pe_angles <- calculateJointAngles(pe, joints)
SummarizedExperiment::assay(pe_angles, "joint_angles")

## End(Not run)
```

`calculateSegmentCOM` *Calculate per-segment centers of mass*

Description

Lower-level function that computes the center of mass for individual body segments from marker position data.

Usage

```
calculateSegmentCOM(pe, proximal_markers, distal_markers, com_fractions)
```

Arguments

`pe`	A `PhysioExperiment` object with `position_x`, `position_y`, and optionally `position_z` assays.

`proximal_markers`
: Character vector of proximal marker names, one per segment.

`distal_markers`
: Character vector of distal marker names, one per segment (same length as `proximal_markers`).

`com_fractions`
: Numeric vector of COM proximal fractions (0-1), one per segment (same length as `proximal_markers`).

Value

A named list of matrices, one per segment. Each matrix has dimensions (n_frames x 2) for 2D data or (n_frames x 3) for 3D data, with columns named `"x"`, `"y"`, and (optionally) `"z"`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`calculateCOM()` for whole-body center of mass, `segmentParameters()` for body segment inertial parameters.

Examples

```
## Not run:
pe <- readTRC("markers.trc")
seg_coms <- calculateSegmentCOM(
  pe,
  proximal_markers = c("RHip", "LHip"),
  distal_markers   = c("RKnee", "LKnee"),
  com_fractions    = c(0.4095, 0.4095)
)

## End(Not run)
```

`calculateStepSymmetry` *Calculate step symmetry from gait parameters*

Description

Computes symmetry metrics comparing left and right sides. Includes the Robinson Symmetry Index and the left/right ratio for each parameter.

Usage

```
calculateStepSymmetry(gait_params)
```

Arguments

gait_params	A gait_parameters object from calculateGaitParameters() containing both left and right sides.

Details

A Symmetry Index (SI) of 0 indicates perfect symmetry. Values above 10 are generally considered clinically asymmetric.

Value

A data.frame with columns:

parameter Name of the gait parameter

left_mean Mean value for the left side

right_mean Mean value for the right side

SI Robinson Symmetry Index: `|L - R| / (0.5 * (L + R)) * 100`

ratio Left / Right ratio

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

calculateGaitParameters() for computing gait temporal-spatial parameters, symmetryIndex() for general symmetry index computation, plotSymmetry() for symmetry visualization.

Examples

```
# See test file for worked examples
```

---

calibrateIMU *Calibrate IMU from static data*

Description

Estimates accelerometer and gyroscope biases from data recorded during a static (motionless) period. The accelerometer bias is computed relative to the expected gravity vector, and the gyroscope bias is the mean angular velocity during the static period.

Usage

```
calibrateIMU(accel_static, gyro_static)
```

Arguments

accel_static	Numeric matrix (n x 3) of accelerometer readings during the static period, in m/s^2.
gyro_static	Numeric matrix (n x 3) of gyroscope readings during the static period, in rad/s.

Details

The function assumes the sensor is stationary. The accelerometer bias is estimated by subtracting the expected gravity contribution from the mean accelerometer reading. Gravity direction is inferred from the mean acceleration direction, and its expected magnitude is 9.81 m/s^2.

The gyroscope bias is simply the mean of the gyroscope readings during the static period (should be near zero for a stationary sensor).

Value

A list with components:

accel_bias Numeric vector of length 3. Mean accelerometer bias for each axis. For a perfectly calibrated sensor at rest, this would be (0, 0, 0) after accounting for gravity.

gyro_bias Numeric vector of length 3. Mean gyroscope bias for each axis (rad/s). A stationary sensor should read zero; any offset is the bias.

References

Madgwick SOH, Harrison AJL, Vaidyanathan R (2011). "Estimation of IMU and MARG orientation using a gradient descent algorithm." IEEE International Conference on Rehabilitation Robotics.

See Also

estimateOrientation(), removeGravity()

Examples

```
# Simulate static IMU data with known biases
n <- 500
accel_bias_true <- c(0.05, -0.03, 0.02)
gyro_bias_true <- c(0.001, -0.002, 0.0015)
accel <- matrix(rep(c(0, 0, -9.81), each = n), ncol = 3) +
  matrix(rep(accel_bias_true, each = n), ncol = 3)
gyro <- matrix(rep(gyro_bias_true, each = n), ncol = 3)
cal <- calibrateIMU(accel, gyro)
```

`cohensD` *Cohen's d Effect Size*

Description

Computes Cohen's d effect size for comparing two groups or conditions, with confidence intervals and qualitative interpretation.

Usage

```
cohensD(x, y, paired = FALSE, pooled = TRUE)
```

Arguments

`x`	Numeric vector for group 1 (or condition 1 if paired).
`y`	Numeric vector for group 2 (or condition 2 if paired).
`paired`	Logical; if TRUE, computes effect size for paired data using the SD of differences as the denominator.
`pooled`	Logical; if TRUE (default), uses pooled SD as denominator. If FALSE, uses the SD of `y` (Glass's delta, treating `y` as control). Ignored when `paired = TRUE`.

Details

For independent groups with `pooled = TRUE`, the pooled SD is:

$$SD_{pooled} = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

For paired data, the denominator is the SD of the within-pair differences.

Interpretation thresholds follow Cohen (1988):

- |d| < 0.2: negligible
- 0.2 <= |d| < 0.5: small
- 0.5 <= |d| < 0.8: medium
- |d| >= 0.8: large

Value

A list with components:

d Cohen's d value

ci_lower Lower bound of 95 percent confidence interval

ci_upper Upper bound of 95 percent confidence interval

interpretation Qualitative label: "negligible", "small", "medium", or "large"

References

Cohen J (1988). Statistical Power Analysis for the Behavioral Sciences. Lawrence Erlbaum Associates.

See Also

`etaSquared()` for ANOVA-based effect sizes, `plotEffectSizeForest()` for forest plot visualization of effect sizes.

Examples

```
set.seed(42)
x <- rnorm(30, mean = 10, sd = 2)
y <- rnorm(30, mean = 8, sd = 2)
result <- cohensD(x, y)
result$d
result$interpretation
```

---

combineTrials *Combine multiple trials with segmented phases*

Description

Combine multiple trials with segmented phases

Usage

```
combineTrials(..., labels = NULL)
```

Arguments

`...`	segmented_phases objects to combine
`labels`	Optional labels for each trial

Value

A multi_trial_phases object

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`segmentPhases()`, `batchNormalize()`, `normalizeMovement()`

`computeAcceleration` *Compute acceleration from position or velocity data*

Description

Computes acceleration (second derivative) from position or velocity assays. If velocity assays already exist, differentiates those (first derivative of velocity). Otherwise, computes the second derivative of position directly.

Usage

```
computeAcceleration(
  pe,
  assay_names = NULL,
  method = "central",
  sampling_rate = NULL
)
```

Arguments

`pe`	A PhysioExperiment object.
`assay_names`	Character vector of assay names to differentiate. If NULL (default), auto-detects velocity_x/y/z (preferred) or position_x/y/z assays.
`method`	Finite difference method: "central" (default), "forward", or "backward".
`sampling_rate`	Sampling rate in Hz. If NULL (default), uses `samplingRate(pe)`.

Value

PhysioExperiment with new accel_x, accel_y, accel_z assays added.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`computeVelocity()` for first-order derivatives, `computeJerk()` for third-order derivatives, `differentiate()` for general-purpose numerical differentiation.

Examples

```
## Not run:
pe <- make_mocap_markers(n_time = 100, n_markers = 4, sr = 120)
pe <- computeAcceleration(pe)

## End(Not run)
```

`computeImpulse` *Compute vertical impulse from GRF*

Description

Integrates vertical force over detected stance phases using trapezoidal integration.

Usage

```
computeImpulse(
  force,
  sampling_rate,
  threshold = 20,
  subtract_threshold = FALSE
)
```

Arguments

`force`	Numeric vector or matrix of vertical force values.
`sampling_rate`	Sampling rate in Hz.
`threshold`	Contact threshold in force units.
`subtract_threshold`	Logical; if `TRUE`, subtracts `threshold` before integrating each stance (negative values clipped to zero).

Value

For vector input: data.frame with one row per stance and columns including `impulse` (force*time units). For matrix input: same structure plus `channel`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`analyzeForcePlate()` for comprehensive force plate analysis, `computeLoadingRate()` for loading rate computation.

Examples

```
grf <- c(rep(0, 50), rep(600, 100), rep(0, 50))
computeImpulse(grf, sampling_rate = 1000)
```

`computeJerk` *Compute jerk from position data*

Description

Computes jerk (third derivative of position) from position assays.

Usage

```
computeJerk(pe, method = "central", sampling_rate = NULL)
```

Arguments

`pe`	A PhysioExperiment object containing position assays.
`method`	Finite difference method: "central" (default), "forward", or "backward".
`sampling_rate`	Sampling rate in Hz. If NULL (default), uses `samplingRate(pe)`.

Value

PhysioExperiment with new jerk_x, jerk_y, jerk_z assays added.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`computeAcceleration()` for second-order derivatives, `computeVelocity()` for first-order derivatives, `differentiate()` for general-purpose numerical differentiation.

Examples

```
## Not run:
pe <- make_mocap_markers(n_time = 100, n_markers = 4, sr = 120)
pe <- computeJerk(pe)

## End(Not run)
```

computeJointPower *Compute joint power from moment and angular velocity*

Description

Compute joint power from moment and angular velocity

Usage

```
computeJointPower(moment, angular_velocity)
```

Arguments

`moment`	Numeric vector of joint moment values.
`angular_velocity`	Numeric vector of joint angular velocity in rad/s.

Value

Numeric vector of joint power (W).

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`inverseDynamics2D()` for computing joint moments in 2D, `inverseDynamics3D()` for computing joint moments in 3D.

Examples

```
computeJointPower(c(10, 12, 8), c(2, 1.5, 1))
```

computeLoadingRate *Compute vertical loading rate from GRF*

Description

Detects stance phases from a vertical GRF signal and computes loading rate from initial contact to peak force for each stance.

Usage

```
computeLoadingRate(
  vgrf,
  sampling_rate,
  threshold = 20,
  method = c("instantaneous", "average")
)
```

Arguments

vgrf	Numeric vector or matrix of vertical GRF values.
sampling_rate	Sampling rate in Hz.
threshold	Contact threshold in force units.
method	Loading-rate method: "instantaneous" (max slope) or "average" (peak rise over time-to-peak).

Value

For vector input: a data.frame with one row per stance. For matrix input: same data.frame with an added channel column.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

analyzeForcePlate() for comprehensive force plate analysis, computeImpulse() for impulse computation.

Examples

```
grf <- c(rep(0, 100), seq(0, 800, length.out = 150),
         seq(800, 0, length.out = 150), rep(0, 100))
computeLoadingRate(grf, sampling_rate = 1000)
```

---

computeRMSEnvelope *Compute moving RMS envelope of EMG*

Description

Compute moving RMS envelope of EMG

Usage

```
computeRMSEnvelope(x, window_samples = 50L, center = TRUE)
```

Arguments

x	Numeric vector or matrix (time x channels).
window_samples	Window length in samples.
center	Logical; if TRUE, uses centered window.

Value

RMS envelope with the same dimensions as x.

References

Merletti R, Parker PA (2004). "Electromyography: Physiology, Engineering, and Non-Invasive Applications." IEEE Press/Wiley.

See Also

rectifyEMG() for signal rectification, normalizeEMG() for MVC or peak normalization, processEMG() for complete EMG processing pipeline.

Examples

```
set.seed(1)
sig <- rnorm(1000)
env <- computeRMSEnvelope(sig, window_samples = 50)
```

---

computeSpeed *Compute scalar speed from velocity*

Description

Computes the magnitude of the velocity vector (Euclidean norm) at each time point for each marker/keypoint.

Usage

```
computeSpeed(pe, velocity_assays = NULL)
```

Arguments

pe	A PhysioExperiment object containing velocity assays.
velocity_assays	Character vector of velocity assay names. If NULL (default), auto-detects velocity_x/y/z or velocity_kp_x/y.

Value

PhysioExperiment with a new "speed" assay containing the scalar speed: sqrt(vx^2 + vy^2 + vz^2).

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`computeVelocity()` for computing velocity components, `computeAcceleration()` for computing acceleration.

Examples

```
## Not run:
pe <- make_mocap_markers(n_time = 100, n_markers = 4, sr = 120)
pe <- computeVelocity(pe)
pe <- computeSpeed(pe)

## End(Not run)
```

---

computeVelocity *Compute velocity from position data*

Description

Computes velocity (first derivative) from position assays in a PhysioExperiment object using finite differences.

Usage

```
computeVelocity(
  pe,
  assay_names = NULL,
  method = "central",
  sampling_rate = NULL
)
```

Arguments

`pe`	A PhysioExperiment object containing position assays.
`assay_names`	Character vector of assay names to differentiate. If NULL (default), auto-detects position_x/y/z or keypoint_x/y assays.
`method`	Finite difference method: "central" (default), "forward", or "backward".
`sampling_rate`	Sampling rate in Hz. If NULL (default), uses `samplingRate(pe)`.

Value

PhysioExperiment with new velocity assays added. For 3D marker data, assays named velocity_x, velocity_y, velocity_z. For 2D keypoint data, assays named velocity_kp_x, velocity_kp_y.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

computeAcceleration() for second-order derivatives, computeSpeed() for scalar speed from velocity components, differentiate() for general-purpose numerical differentiation.

Examples

```
## Not run:
pe <- make_mocap_markers(n_time = 100, n_markers = 4, sr = 120)
pe <- computeVelocity(pe)
# velocity_x, velocity_y, velocity_z assays are now present

## End(Not run)
```

---

correctSwaps *Correct Marker Label Swaps*

Description

Repairs marker trajectories by swapping the data columns of detected swap pairs from the swap frame onward.

Usage

```
correctSwaps(pe, swaps, assay_prefix = "position", min_confidence = 0.5)
```

Arguments

pe	A PhysioExperiment with position assays.
swaps	A data.frame as returned by detectSwaps(), containing at minimum frame, marker_a, marker_b, and confidence columns.
assay_prefix	Prefix for position assay names. Default "position".
min_confidence	Minimum confidence threshold for applying a correction. Swaps below this threshold are skipped. Default 0.5.

Value

A PhysioExperiment with corrected position assays.

See Also

detectSwaps(), trackMarkers()

Examples

```
## Not run:
swaps <- detectSwaps(pe)
pe_fixed <- correctSwaps(pe, swaps, min_confidence = 0.3)

## End(Not run)
```

`create_schema_from_opensim` *Create a TaskSchema from OpenSim model for gait analysis*

Description

Generates a TaskSchema based on an OpenSim model's markers and available signal channels. This function creates appropriate event definitions that can be used with `detectEvents()` for automatic event detection.

Usage

```
create_schema_from_opensim(
  model,
  task_type = c("gait", "running", "jump", "generic"),
  available_signals = NULL,
  side = "right"
)
```

Arguments

`model`	One of: • an OpenSim model object from signalIO's `read_osim()`, • a path to an `.osim` model file, or • a model summary list from `PhysioOpenSim::opensimModelSummary()`.
`task_type`	Type of movement task: "gait", "running", "jump", "generic"
`available_signals`	Character vector of signal names available in the data (e.g., from IK, ID, or GRF data)
`side`	Character, which side to base events on: "right", "left", or "both"

Details

This function analyzes the model's markers and the available signals to generate an appropriate TaskSchema. For gait analysis, it prefers GRF-based detection if available, falling back to marker-based or kinematic detection.

Value

A TaskSchema object configured for the specified task type

References

Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, Guendelman E, Thelen DG (2007). "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement." IEEE Transactions on Biomedical Engineering, 54(11), 1940-1950.

See Also

`batch_analyze_opensim()` for batch processing of OpenSim sessions, `detectEvents()` for event detection using task schemas.

Examples

```
## Not run:
library(signalIO)

# Load OpenSim model
model <- read_osim("gait2354.osim")

# Load IK results
ik <- read_mot("ik_results.mot")
ik_channels <- rownames(ik)

# Create gait schema with available channels
schema <- create_schema_from_opensim(model, "gait", ik_channels)
print(schema)

# Use schema for event detection
events <- detectEvents(data, schema)

## End(Not run)
```

`createBenchmarkExample` *Create a synthetic benchmark dataset bundle*

Description

Generates prediction/reference CSV pairs and a manifest for dry-run validation benchmarking when external data are not yet available.

Usage

```
createBenchmarkExample(
  output_dir = tempdir(),
  n_trials = 3L,
  n_samples = 300L,
  variables = c("hip_angle", "knee_angle", "ankle_angle"),
  noise_sd = 0.02,
  seed = 1
)
```

Arguments

`output_dir`	Output directory where files are written.
`n_trials`	Number of synthetic trials.
`n_samples`	Samples per trial.
`variables`	Character vector of variable names.
`noise_sd`	Prediction noise SD relative to the generated reference.
`seed`	Random seed.

Value

A list with `manifest`, `manifest_path`, and `data_dir`.

References

Bland JM, Altman DG (1986). "Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement." Lancet, 327(8476), 307-310.

See Also

`runBenchmarkSuite()` for running the benchmark suite, `benchmarkManifestTemplate()` for creating empty manifest templates.

Examples

```
ex <- createBenchmarkExample(n_trials = 2, seed = 1)
runBenchmarkSuite(ex$manifest, data_dir = ex$data_dir)
```

defaultBenchmarkThresholds *Get default benchmark thresholds*

Description

Returns a named list of threshold values used by benchmark pass/fail logic.

Usage

```
defaultBenchmarkThresholds(profile = c("balanced", "strict", "lenient"))
```

Arguments

profile	Threshold profile: "balanced" (default), "strict", or "lenient".

Value

Named list with threshold fields: rmse_max, mae_max, bias_abs_max, cor_min, icc_min.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

See Also

benchmarkAgreement() for computing agreement metrics, runBenchmarkSuite() for running a full benchmark suite.

Examples

```
defaultBenchmarkThresholds()
```

define_skeleton *Create a pre-defined skeleton model*

Description

Factory function that returns a SkeletonModel for a well-known pose estimation or motion capture marker set.

Usage

```
define_skeleton(model_name)
```

Arguments

model_name	Character string. One of:
	"BODY_25" OpenPose BODY_25 model (25 keypoints, 24 bones).
	"COCO" OpenPose COCO model (18 keypoints, 17 bones).
	"BlazePose" MediaPipe BlazePose model (33 keypoints, 35 bones).
	"PluginGait" Vicon Plug-in Gait full-body marker set (39 markers, 38 bones).

Value

A SkeletonModel object.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

SkeletonModel(), get_bone_connections(), get_limb_pairs()

Examples

```
sk <- define_skeleton("BODY_25")
print(sk)

sk_coco <- define_skeleton("COCO")
get_bone_connections(sk_coco)
```

---

demoMoCapData *Create a beginner-friendly demo dataset*

Description

Generates synthetic motion capture, force, and EMG data so first-time users can run package workflows without external files.

Usage

```
demoMoCapData(
  n_frames = 300,
  sampling_rate = 120,
  n_markers = 8,
  emg_sampling_rate = 1000,
  seed = 123
)
```

Arguments

n_frames	Number of MoCap frames to generate.
sampling_rate	MoCap sampling rate in Hz.
n_markers	Number of markers in the synthetic marker set.
emg_sampling_rate	EMG sampling rate in Hz.
seed	Optional random seed for reproducibility. Set to NULL to skip setting a seed.

Value

A named list with:

mocap A PhysioExperiment with position_x, position_y, position_z assays.

grf Numeric vector of synthetic vertical GRF.

forces Matrix with force_x, force_y, force_z columns.

joints Data frame with 2D joint-center coordinates for ankle, knee, and hip.

joint_angles Data frame with ankle/knee/hip joint angles (radians).

emg Matrix of synthetic EMG channels.

sampling_rate MoCap sampling rate (Hz).

emg_sampling_rate EMG sampling rate (Hz).

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

quickStartMoCap() for a complete getting-started workflow, assessMoCapReadiness() for data quality assessment.

Examples

```
demo <- demoMoCapData(seed = 1)
demo$mocap
head(demo$joints)
```

`detectEvents` *Detect events in movement data*

Description

Automatically detects events defined in a TaskSchema based on signal data.

Usage

```
detectEvents(
  x,
  schema,
  signals = NULL,
  method = c("auto", "manual", "hybrid"),
  sampling_rate = NULL,
  ...
)
```

Arguments

`x`	PhysioExperiment object or matrix (time x channels)
`schema`	TaskSchema object defining events to detect
`signals`	Named list of signal vectors for detection. If NULL and x is a PhysioExperiment, signals are extracted from column names. If x is a matrix with named columns, signals are auto-created from those names; otherwise provide `signals` explicitly.
`method`	Detection approach: • "auto" - Automatic detection using schema definitions • "manual" - Use typical_timing from schema as event times • "hybrid" - Try auto first, fall back to typical_timing if failed
`sampling_rate`	Sampling rate in Hz. Required if x is a matrix.
`...`	Additional arguments passed to detection methods

Value

A data.frame with columns:

- event - Event name
- label - Human-readable label
- index - Sample index of event
- time - Time in seconds
- percent - Percent of movement (if applicable)
- method - Detection method used
- confidence - Detection confidence (0-1)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`manualEvents()`, `segmentPhases()`, `TaskSchema()`

Examples

```
# Create synthetic gait data
set.seed(123)
n <- 1000
t <- seq(0, 1, length.out = n)
vGRF <- c(rep(0, 100), sin(seq(0, pi, length.out = 500)) * 800, rep(0, 400))
vGRF <- vGRF + rnorm(n, 0, 10)

signals <- list(vGRF = vGRF)
events <- detectEvents(as.matrix(vGRF), schema_gait,
                       signals = signals, sampling_rate = 1000)
```

---

detectForcePlateContacts

Detect contact windows for one or more force plates

Description

Identifies threshold-based contact windows from vertical GRF signals.

Usage

```
detectForcePlateContacts(vgrf, threshold = 20, sampling_rate = NULL)
```

Arguments

`vgrf`	Numeric vector or matrix of vertical GRF (rows = time, columns = plates/channels).
`threshold`	Contact threshold in force units.
`sampling_rate`	Optional sampling rate in Hz. If provided, durations are returned in seconds.

Value

A data.frame with one row per detected contact and columns: `plate`, `stance`, `onset`, `offset`, `duration_samples`, `duration_s`, and `peak_force`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

analyzeForcePlate() for comprehensive force plate analysis, analyzeForcePlatePE() for PhysioExperiment-based analysis.

Examples

```
v <- cbind(
  fp1 = c(rep(0, 100), rep(500, 200), rep(0, 100)),
  fp2 = c(rep(0, 200), rep(700, 100), rep(0, 100))
)
detectForcePlateContacts(v, threshold = 20, sampling_rate = 1000)
```

---

detectGaps *Detect gaps (contiguous NA runs) in assay data*

Description

Scans the specified assay of a PhysioExperiment object for contiguous runs of NA values and returns a data.frame describing each gap.

Usage

```
detectGaps(pe, assay_name = "position_x")
```

Arguments

pe	A PhysioExperiment object
assay_name	Name of the assay to scan (default: "position_x")

Value

A data.frame with columns:

- channel - Channel (column) name or index
- start - Start index of the gap (1-based)
- end - End index of the gap (1-based)
- size - Number of consecutive NA values

References

Federolf PA (2013). "A novel approach to solve the 'missing marker problem' in marker-based motion analysis that does not require additional assumptions about the biodynamic model." Journal of Biomechanics, 46(13), 2173-2178.

See Also

reportGaps(), fillGaps(), fillGapsLinear()

Examples

```
pe <- PhysioCore::PhysioExperiment(
  assays = S4Vectors::SimpleList(position_x = matrix(c(1, NA, NA, 4), ncol = 1)),
  colData = S4Vectors::DataFrame(label = "M1", type = "marker"),
  samplingRate = 120
)
gaps <- detectGaps(pe, assay_name = "position_x")
```

---

detectSwaps *Detect Marker Label Swaps*

Description

Identifies frames where two or more marker trajectories abruptly swap positions, indicating a labelling error. Swaps are detected by velocity spikes that exceed `median + 5 * MAD` and show a cross-over pattern between marker pairs.

Usage

```
detectSwaps(pe, threshold = NULL, window = 5, assay_prefix = "position")
```

Arguments

`pe`	A PhysioExperiment with position assays.
`threshold`	Velocity threshold for spike detection. If `NULL` (default), computed as `median + 5 * MAD` of each marker's velocity.
`window`	Number of frames around a spike to check for cross-over patterns. Default 5.
`assay_prefix`	Prefix for position assay names. Default `"position"`.

Value

A `data.frame` with columns:

frame Frame index where the swap was detected.

marker_a Name or index of the first marker in the swap pair.

marker_b Name or index of the second marker.

velocity_a Velocity of marker_a at the swap frame.

velocity_b Velocity of marker_b at the swap frame.

confidence Confidence score (0-1) based on how closely the cross-over distances match.

If no swaps are detected, an empty data.frame with the same columns is returned.

See Also

`trackMarkers()`, `correctSwaps()`

Examples

```
## Not run:
pe <- readVenus3D("capture.csv")
pe_tracked <- trackMarkers(pe)
swaps <- detectSwaps(pe_tracked)
if (nrow(swaps) > 0) {
  pe_fixed <- correctSwaps(pe_tracked, swaps)
}

## End(Not run)
```

`differentiate` *Differentiate a numeric vector or matrix*

Description

Low-level function to compute numerical derivatives of arbitrary order using finite difference methods.

Usage

```
differentiate(x, dt, method = c("central", "forward", "backward"), order = 1L)
```

Arguments

`x`	Numeric vector or matrix (time x channels).
`dt`	Time step between samples (1 / sampling_rate).
`method`	Difference method: "central", "forward", or "backward".
`order`	Derivative order: 1 (velocity), 2 (acceleration), or 3 (jerk).

Details

For `method = "central"` with `order = 1`:

$$f'(i) = (x[i+1] - x[i-1])/(2 \cdot dt)$$

For `method = "central"` with `order = 2`:

$$f''(i) = (x[i+1] - 2x[i] + x[i-1])/dt^2$$

For higher orders, the derivative is computed by repeated application of the first-order formula.

Value

Differentiated data with same dimensions as input. Boundary values where the stencil cannot be applied are set to NA.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`computeVelocity()` for computing velocity from PhysioExperiment, `computeAcceleration()` for second-order derivatives, `savgolFilter()` for smoothed differentiation via Savitzky-Golay.

Examples

```
# Differentiate sin to get cos
t <- seq(0, 2 * pi, length.out = 200)
x <- sin(t)
dx <- differentiate(x, dt = t[2] - t[1], method = "central", order = 1)
# dx should approximate cos(t)
```

---

dtwAverage *DTW Barycenter Averaging (DBA)*

Description

Computes the average waveform using DTW Barycenter Averaging, which accounts for time warping in the averaging process.

Usage

```
dtwAverage(x, init = "medoid", max_iter = 30, tol = 1e-04, window_size = NULL)
```

Arguments

`x`	A PhysioExperiment object or matrix (time x observations).
`init`	Initial reference: "medoid", "mean", or a numeric vector.
`max_iter`	Maximum iterations for DBA.
`tol`	Convergence tolerance.
`window_size`	Sakoe-Chiba band width.

Value

A list containing:

`average`	The DTW-averaged waveform
`iterations`	Number of iterations used
`alignments`	List of alignment paths to the average

References

Sakoe H, Chiba S (1978). "Dynamic programming algorithm optimization for spoken word recognition." IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49.

Petitjean F, Ketterlin A, Gancarski P (2011). "A global averaging method for dynamic time warping, with applications to clustering." Pattern Recognition, 44(3), 678-693.

See Also

dtwDistance(), dtwDistanceMatrix(), dtwClustering()

Examples

```
# Average multiple gait cycles with timing variation
set.seed(123)
t <- seq(0, 100, length.out = 100)
data <- sapply(1:20, function(i) {
  phase <- rnorm(1, 0, 10)
  sin(2 * pi * (t + phase) / 100) * 30 + rnorm(100, 0, 2)
})

avg <- dtwAverage(data)
plot(avg$average, type = "l")
```

---

dtwClustering *DTW-based clustering*

Description

Clusters waveforms using DTW distance with hierarchical or k-medoids method.

Usage

```
dtwClustering(
  x,
  k = 2,
  method = c("hierarchical", "kmedoids"),
  linkage = "ward.D2",
  window_size = NULL
)
```

Arguments

x	A PhysioExperiment object or matrix (time x observations).
k	Number of clusters.
method	Clustering method: "hierarchical" or "kmedoids".
linkage	For hierarchical: "ward.D2", "complete", "average", etc.
window_size	Sakoe-Chiba band width.

Value

A list containing:

`clusters`	Cluster assignments
`centers`	Cluster centers (medoids or DBA averages)
`distance_matrix`	DTW distance matrix used

References

Sakoe H, Chiba S (1978). "Dynamic programming algorithm optimization for spoken word recognition." IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49.

See Also

`dtwDistance()`, `dtwDistanceMatrix()`, `dtwAverage()`

Examples

```
# Cluster gait patterns
set.seed(123)
t <- seq(0, 100, length.out = 100)

# Generate two groups with different patterns
group1 <- sapply(1:15, function(i) {
  sin(2 * pi * t / 100) * 30 + rnorm(100, 0, 3)
})
group2 <- sapply(1:15, function(i) {
  sin(2 * pi * t / 100 + pi/4) * 20 + rnorm(100, 0, 3)
})
data <- cbind(group1, group2)

result <- dtwClustering(data, k = 2)
table(result$clusters)
```

`dtwDistance` *Dynamic Time Warping (DTW) for Biomechanics*

Description

Functions for Dynamic Time Warping analysis including distance computation, alignment, averaging, and clustering of biomechanical waveforms. Compute DTW distance between two time series

Usage

```
dtwDistance(
  x,
  y,
  window_size = NULL,
  step_pattern = c("symmetric2", "symmetric1", "asymmetric"),
  normalize = TRUE
)
```

Arguments

x	First time series (numeric vector or matrix column).
y	Second time series (numeric vector or matrix column).
window_size	Sakoe-Chiba band width (NULL for no constraint).
step_pattern	Step pattern: "symmetric1", "symmetric2", or "asymmetric".
normalize	Logical; whether to normalize distance by path length.

Details

Calculates the Dynamic Time Warping distance and alignment path between two waveforms, allowing for non-linear time alignment.

DTW finds the optimal alignment between two time series by warping the time axis. It's useful for comparing movements that may occur at different speeds or with timing differences.

Value

A list of class "dtw_result" containing:

distance	DTW distance
normalized_distance	Distance normalized by path length
path	Alignment path (matrix with columns 'index1', 'index2')
cost_matrix	Accumulated cost matrix

References

Sakoe H, Chiba S (1978). "Dynamic programming algorithm optimization for spoken word recognition." IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49.

See Also

dtwDistanceMatrix(), dtwAverage(), dtwClustering(), dtwWarp()

Examples

```
# Two gait cycles with different timing
t <- seq(0, 100, length.out = 100)
x <- sin(2 * pi * t / 100) * 30
y <- sin(2 * pi * (t + 10) / 100) * 30  # Phase shifted

result <- dtwDistance(x, y)
print(result$distance)
```

---

dtwDistanceMatrix *Compute DTW distance matrix*

Description

Computes pairwise DTW distances between all columns of a matrix or between observations in a PhysioExperiment.

Usage

```
dtwDistanceMatrix(x, window_size = NULL, normalize = TRUE, parallel = FALSE)
```

Arguments

x	A PhysioExperiment object or matrix (time x observations).
window_size	Sakoe-Chiba band width constraint.
normalize	Logical; normalize by path length.
parallel	Logical; use parallel processing.

Value

A symmetric distance matrix.

References

Sakoe H, Chiba S (1978). "Dynamic programming algorithm optimization for spoken word recognition." IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49.

See Also

dtwDistance(), dtwAverage(), dtwClustering()

Examples

```
# Create matrix of 10 gait cycles
set.seed(123)
t <- seq(0, 100, length.out = 100)
data <- sapply(1:10, function(i) {
  phase <- rnorm(1, 0, 10)
  sin(2 * pi * (t + phase) / 100) * 30 + rnorm(100, 0, 2)
})

dist_matrix <- dtwDistanceMatrix(data)
```

---

dtwWarp *Warp a waveform using DTW alignment*

Description

Applies a DTW alignment path to warp one waveform to match another.

Usage

```
dtwWarp(x, dtw_result, to = c("query", "reference"))
```

Arguments

x	Waveform to warp.
dtw_result	DTW result from dtwDistance().
to	Which series to warp to: "query" (index1) or "reference" (index2).

Value

Warped waveform.

References

Sakoe H, Chiba S (1978). "Dynamic programming algorithm optimization for spoken word recognition." IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49.

See Also

dtwDistance(), dtwAverage(), normalizeMovement()

`estimateOrientation` *Estimate orientation from IMU sensors using sensor fusion*

Description

Fuses accelerometer, gyroscope, and optionally magnetometer data to estimate 3D orientation over time. Returns both quaternion and Euler angle representations.

Usage

```
estimateOrientation(
  accel,
  gyro,
  mag = NULL,
  method = c("madgwick", "complementary"),
  beta = 0.1,
  sampling_rate
)
```

Arguments

`accel`	Numeric matrix (n x 3) of accelerometer readings in m/s^2. Columns correspond to x, y, z axes.
`gyro`	Numeric matrix (n x 3) of gyroscope readings in rad/s. Columns correspond to x, y, z axes.
`mag`	Numeric matrix (n x 3) of magnetometer readings, or `NULL` if unavailable. Default `NULL`.
`method`	Character. Sensor fusion method: `"madgwick"` (default) or `"complementary"`.
`beta`	Numeric. Filter gain for the Madgwick filter (default 0.1). For the complementary filter this parameter sets the alpha coefficient (gyroscope trust weight). Lower values for Madgwick or higher values for complementary increase reliance on the gyroscope.
`sampling_rate`	Numeric. Sampling rate in Hz.

Details

The Madgwick filter uses a gradient-descent algorithm to correct gyroscope drift using accelerometer (and optionally magnetometer) measurements. The `beta` parameter controls the correction strength.

The complementary filter blends gyroscope integration (high-pass) with accelerometer tilt estimation (low-pass). The `beta` parameter serves as the alpha coefficient, controlling the balance between gyroscope and accelerometer contributions.

Value

A data.frame with columns:

time Time in seconds from start.

roll Roll angle (degrees), rotation about X axis.

pitch Pitch angle (degrees), rotation about Y axis.

yaw Yaw angle (degrees), rotation about Z axis.

q_w Quaternion scalar component.

q_x Quaternion x component.

q_y Quaternion y component.

q_z Quaternion z component.

References

Madgwick SOH, Harrison AJL, Vaidyanathan R (2011). "Estimation of IMU and MARG orientation using a gradient descent algorithm." IEEE International Conference on Rehabilitation Robotics.

See Also

`removeGravity()`, `calibrateIMU()`, `quaternionToEuler()`

Examples

```
# Simulate static IMU data (sensor resting with gravity along -Z)
n <- 100
accel <- matrix(c(rep(0, n), rep(0, n), rep(-9.81, n)), ncol = 3)
gyro <- matrix(0, nrow = n, ncol = 3)
result <- estimateOrientation(accel, gyro, sampling_rate = 100)
head(result)
```

---

estimateSegmentInertia

Estimate segment inertial properties for lower-limb inverse dynamics

---

Description

Builds foot/shank/thigh inertial parameters from body mass and segment lengths using De Leva-style coefficients.

Usage

```
estimateSegmentInertia(
  body_mass,
  segment_lengths = NULL,
  body_height = NULL,
  model = c("deLeva_male", "deLeva_female")
)
```

Arguments

body_mass	Body mass in kilograms.
segment_lengths	Named numeric vector with foot, shank, and thigh lengths in meters. If NULL, lengths are estimated from body_height.
body_height	Body height in meters, required when segment_lengths = NULL.
model	Anthropometric coefficient set: "deLeva_male" or "deLeva_female".

Value

A data.frame with segment mass, COM fraction, radius of gyration, and segment moment of inertia about COM.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

segmentParameters() for body segment inertial parameters, inverseDynamics2D() for 2D inverse dynamics computation, inverseDynamics3D() for 3D inverse dynamics computation.

Examples

```
inertia <- estimateSegmentInertia(
  body_mass = 70,
  segment_lengths = c(foot = 0.25, shank = 0.43, thigh = 0.45)
)
```

---

etaSquared *Eta-Squared Effect Size*

Description

Computes eta-squared, partial eta-squared, and omega-squared from a one-way between-subjects comparison.

Usage

```
etaSquared(x, groups)
```

Arguments

x	Numeric vector of values.
groups	Factor or character vector of group membership.

Details

Eta-squared is the proportion of total variance explained by group membership:

$$\eta^2 = \frac{SS_{between}}{SS_{total}}$$

Omega-squared provides a less biased estimate:

$$\omega^2 = \frac{SS_{between} - df_{between} \cdot MS_{within}}{SS_{total} + MS_{within}}$$

For one-way designs, partial eta-squared equals eta-squared.

Value

A list with components:

eta_sq Eta-squared (SS_between / SS_total)

partial_eta_sq Partial eta-squared (SS_between / (SS_between + SS_within))

omega_sq Omega-squared (bias-corrected effect size)

References

Cohen J (1988). "Statistical Power Analysis for the Behavioral Sciences." Lawrence Erlbaum Associates.

See Also

`cohensD()` for pairwise effect sizes, `plotEffectSizeForest()` for forest plot visualization.

Examples

```
set.seed(42)
x <- c(rnorm(20, 10, 2), rnorm(20, 12, 2), rnorm(20, 14, 2))
groups <- rep(c("A", "B", "C"), each = 20)
result <- etaSquared(x, groups)
result$eta_sq
```

---

eulerToQuaternion *Convert Euler angles to quaternion*

Description

Converts Euler angles (roll, pitch, yaw) to quaternion (w, x, y, z) representation using the specified rotation order.

Usage

```
eulerToQuaternion(roll, pitch, yaw, order = "ZYX")
```

Arguments

`roll`	Numeric. Rotation about X axis (in radians by default, or degrees if values > 2*pi suggest degree input).
`pitch`	Numeric. Rotation about Y axis.
`yaw`	Numeric. Rotation about Z axis.
`order`	Character. Rotation order. Default `"ZYX"`.

Details

Input angles are assumed to be in **radians**. For the `"ZYX"` convention, the combined quaternion is computed as: $q = q_z \otimes q_y \otimes q_x$

The resulting quaternion is always returned with $w \geq 0$ (canonical form).

Value

A matrix with columns `w`, `x`, `y`, `z`. If inputs are scalars, returns a 1-row matrix. If inputs are vectors, returns a matrix with one row per element.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

Grood ES, Suntay WJ (1983). "A joint coordinate system for the clinical description of three-dimensional motions: application to the knee." Journal of Biomechanical Engineering, 105(2), 136-144.

See Also

`quaternionToEuler()`, `calculateJointAngles()`, `vectorAngle()`

Examples

```
# Zero rotation -> identity quaternion
eulerToQuaternion(0, 0, 0)

# 90-degree rotation about Z axis (input in radians)
eulerToQuaternion(0, 0, pi/2)
```

`Event` *Create an Event definition*

Description

Defines an event within a movement task that can be detected automatically or specified manually.

Usage

```
Event(
  name,
  label,
  detection_method = "threshold",
  detection_params = list(),
  typical_timing = NULL
)
```

Arguments

`name`	Short identifier for the event (e.g., "hs1", "to")
`label`	Human-readable label (e.g., "Heel Strike", "Toe Off")
`detection_method`	Method for automatic detection: • "threshold" - Signal crosses a threshold value • "peak" - Local maximum or minimum • "zero_crossing" - Signal crosses zero • "angle" - Specific angle value (for cyclic data) • "velocity_threshold" - Velocity exceeds threshold • "manual" - User-specified timing
`detection_params`	List of parameters for detection method: • For "threshold": signal, threshold, direction ("rising"/"falling") • For "peak": signal, type ("max"/"min"), prominence • For "zero_crossing": signal, direction ("rising"/"falling"/"any") • For "angle": signal, value
`typical_timing`	Expected timing as percentage of movement (0-100), used for validation and as fallback

Value

An Event object (S3 class)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`Phase()`, `TaskSchema()`, `detectEvents()`

Examples

```
# Heel strike event detected when vertical GRF rises above 10N
hs <- Event("hs", "Heel Strike", "threshold",
            list(signal = "vGRF", threshold = 10, direction = "rising"),
            typical_timing = 0)

# Toe off event detected when vertical GRF falls below 10N
to <- Event("to", "Toe Off", "threshold",
            list(signal = "vGRF", threshold = 10, direction = "falling"),
            typical_timing = 60)

# Peak knee flexion
pk <- Event("peak_flex", "Peak Knee Flexion", "peak",
            list(signal = "knee_angle", type = "max"))
```

`extractPhase` *Extract a single phase from segmented data*

Description

Extract a single phase from segmented data

Usage

```
extractPhase(x, phase_name, search_subphases = TRUE)
```

Arguments

`x`	A segmented_phases object
`phase_name`	Name of the phase to extract
`search_subphases`	Whether to search within subphases

Value

A matrix of phase data

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`segmentPhases()`, `phaseTiming()`, `getPhaseData()`

`extractWaveformFeatures` *Dimensionality Reduction for Biomechanics*

Description

Functions for dimensionality reduction and visualization of high-dimensional biomechanical waveform data including PCA, UMAP, and t-SNE. Extract waveform features for dimensionality reduction

Usage

```
extractWaveformFeatures(x, features = c("statistical", "shape"), n_points = 50)
```

Arguments

`x`	A PhysioExperiment object or matrix (time x observations).
`features`	Character vector of features to extract: "raw" (flattened waveform), "statistical" (summary stats), "frequency" (spectral features), "shape" (curve characteristics).
`n_points`	For "raw", number of points to resample to.

Details

Extracts summary features from waveforms for use with standard dimensionality reduction methods.

Feature types:

- raw: The raw waveform resampled to n_points
- statistical: Mean, SD, min, max, range, skewness, kurtosis
- frequency: Dominant frequency, spectral centroid, bandwidth
- shape: Peaks, zero crossings, area under curve

Value

A matrix (observations x features) suitable for PCA/UMAP.

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

`waveformPCA()`, `waveformUMAP()`, `waveformTSNE()`

Examples

```
# Extract features from gait data
set.seed(123)
data <- matrix(rnorm(1000), nrow = 100, ncol = 10)
features <- extractWaveformFeatures(data, features = c("statistical", "shape"))
```

`fillGaps` *Fill gaps in motion capture data*

Description

Fills NA gaps in one or more position assays of a PhysioExperiment object using interpolation. Gaps larger than `max_gap` are left as NA.

Usage

```
fillGaps(pe, method = c("spline", "linear"), max_gap = 50, assay_names = NULL)
```

Arguments

`pe`	A PhysioExperiment object
`method`	Interpolation method: `"linear"` uses `approx`, `"spline"` uses `smooth.spline`
`max_gap`	Maximum gap size (in samples) to fill. Gaps larger than this are left as NA. Set to `Inf` to fill all gaps.
`assay_names`	Character vector of assay names to fill. If `NULL` (default), fills all position assays that are present (position_x, position_y, position_z, keypoint_x, keypoint_y).

Value

PhysioExperiment with filled assays (modified in place)

References

Federolf PA (2013). "A novel approach to solve the 'missing marker problem' in marker-based motion analysis that does not require additional assumptions about the biodynamic model." Journal of Biomechanics, 46(13), 2173-2178.

See Also

`detectGaps()`, `reportGaps()`, `fillGapsLinear()`, `fillGapsSpline()`

Examples

```
pe <- PhysioCore::PhysioExperiment(
  assays = S4Vectors::SimpleList(
    position_x = matrix(c(1, NA, NA, 4, 5), ncol = 1),
    position_y = matrix(c(10, NA, NA, 40, 50), ncol = 1)
  ),
  colData = S4Vectors::DataFrame(label = "M1", type = "marker"),
  samplingRate = 120
)
pe_filled <- fillGaps(pe, method = "linear", max_gap = 50)
```

---

fillGapsLinear *Linear interpolation for a single vector*

Description

Fills NA values in a numeric vector using linear interpolation via approx.

Usage

```
fillGapsLinear(x)
```

Arguments

x	Numeric vector potentially containing NAs

Value

Numeric vector with NAs filled by linear interpolation. Leading and trailing NAs (outside the range of valid data) remain as NA.

References

Federolf PA (2013). "A novel approach to solve the 'missing marker problem' in marker-based motion analysis that does not require additional assumptions about the biodynamic model." Journal of Biomechanics, 46(13), 2173-2178.

See Also

fillGapsSpline(), fillGaps(), detectGaps()

Examples

```
x <- c(1, NA, NA, 4, 5, NA, 7)
fillGapsLinear(x)
```

`fillGapsSpline` *Spline interpolation for a single vector*

Description

Fills NA values in a numeric vector using smooth spline interpolation via `smooth.spline`.

Usage

```
fillGapsSpline(x, spar = NULL)
```

Arguments

`x`	Numeric vector potentially containing NAs
`spar`	Smoothing parameter passed to `smooth.spline`. If `NULL` (default), the smoothing parameter is chosen automatically.

Value

Numeric vector with NAs filled by spline interpolation. Leading and trailing NAs (outside the range of valid data) remain as NA.

References

Federolf PA (2013). "A novel approach to solve the 'missing marker problem' in marker-based motion analysis that does not require additional assumptions about the biodynamic model." Journal of Biomechanics, 46(13), 2173-2178.

See Also

`fillGapsLinear()`, `fillGaps()`, `detectGaps()`

Examples

```
x <- c(1, NA, NA, 4, 5, NA, 7)
fillGapsSpline(x)
```

`filterGRF` *Low-pass filter ground reaction force data*

Description

Applies low-pass filtering to force-plate signals. By default this uses a zero-phase Butterworth filter (requires the `signal` package). If `signal` is not available, the function falls back to a moving-average filter.

Usage

```
filterGRF(
  x,
  sampling_rate,
  cutoff = 20,
  order = 4,
  method = c("butterworth", "moving_average")
)
```

Arguments

`x`	Numeric vector or matrix (time x channels).
`sampling_rate`	Sampling rate in Hz.
`cutoff`	Low-pass cutoff frequency in Hz.
`order`	Butterworth filter order.
`method`	Filtering method: `"butterworth"` or `"moving_average"`.

Value

Filtered data with the same dimensions as `x`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`butterworthFilter()` for general Butterworth filtering, `analyzeForcePlate()` for comprehensive force plate analysis.

Examples

```
sr <- 1000
t <- seq(0, 1, length.out = sr)
x <- sin(2 * pi * 10 * t) + 0.2 * sin(2 * pi * 150 * t)
y <- filterGRF(x, sampling_rate = sr, cutoff = 25)
```

`filterSignals` *Filter signals in a PhysioExperiment object*

Description

Applies a Butterworth IIR filter to signal data stored in a PhysioExperiment object. The filter is applied using zero-phase filtering (forward-backward) to avoid phase distortion.

Usage

```
filterSignals(
  pe,
  type = c("lowpass", "highpass", "bandpass", "bandstop"),
  cutoff,
  order = 4,
  assay_name = NULL,
  output_assay = NULL
)
```

Arguments

`pe`	A PhysioExperiment object.
`type`	Filter type: "lowpass", "highpass", "bandpass", or "bandstop".
`cutoff`	Cutoff frequency in Hz. A single value for "lowpass" or "highpass"; a length-2 vector `c(low, high)` for "bandpass" or "bandstop".
`order`	Filter order (default: 4).
`assay_name`	Which assay to filter. If NULL, uses the first assay.
`output_assay`	Name for the output assay. If NULL, defaults to `"{assay_name}_filtered"`.

Details

This function requires the **signal** package (listed in Suggests). If not installed, an informative error message with install instructions is provided.

The Butterworth filter is designed using `signal::butter()` and applied with `signal::filtfilt()` for zero-phase filtering. The cutoff frequency is normalized to the Nyquist frequency automatically.

Value

A PhysioExperiment object with filtered data stored as a new assay.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

Butterworth S (1930). "On the Theory of Filter Amplifiers." Wireless Engineer, 7, 536-541.

See Also

butterworthFilter() for filtering raw vectors and matrices, savgolFilter() for Savitzky-Golay polynomial smoothing, movingAverage() for simple moving average smoothing.

Examples

```
## Not run:
pe <- make_mocap_markers(n_time = 500, n_markers = 4, sr = 120)
pe_filt <- filterSignals(pe, type = "lowpass", cutoff = 10)

## End(Not run)
```

---

fPCA *Functional Data Analysis (FDA) for Biomechanics*

Description

Functions for functional data analysis including functional PCA (fPCA), functional regression, and curve registration. These methods treat biomechanical waveforms as continuous functions. Functional Principal Component Analysis (fPCA)

Usage

```
fPCA(x, n_components = 5, smooth = FALSE, smooth_param = 10)
```

Arguments

x	A PhysioExperiment object or matrix (time x observations).
n_components	Number of principal components to retain.
smooth	Logical; if TRUE, smooths the data before analysis.
smooth_param	Smoothing parameter (higher = smoother).

Details

Performs functional PCA on waveform data to identify the main modes of variation in movement patterns.

fPCA decomposes waveform variability into orthogonal modes. In gait analysis, PC1 often represents overall amplitude, PC2 timing/phase shifts, and subsequent PCs capture more subtle shape variations.

Value

A list of class "fpca_result" containing:

`scores`	PC scores for each observation (observations x components)
`loadings`	PC loadings/eigenfunctions (time x components)
`variance_explained`	Proportion of variance explained by each PC
`cumulative_variance`	Cumulative variance explained
`mean_function`	Mean waveform across observations

References

Ramsay JO, Silverman BW (2005). "Functional Data Analysis." 2nd ed. Springer.

See Also

`reconstructFPCA()` for waveform reconstruction from fPCA results, `plotFPCA()` for visualization of fPCA results, `registerCurves()` for separating phase and amplitude variation.

Examples

```
# Simulate gait angle data (100 time points x 30 subjects)
set.seed(123)
t <- seq(0, 100, length.out = 100)
base_curve <- sin(2 * pi * t / 100) * 30

# Add subject variability
data <- sapply(1:30, function(i) {
  amplitude <- rnorm(1, 1, 0.2)
  phase <- rnorm(1, 0, 5)
  base_curve * amplitude + rnorm(100, 0, 2)
})

pe <- PhysioExperiment(assays = list(values = data), samplingRate = 100)
fpca_result <- fPCA(pe, n_components = 4)

# Plot first two PC loadings
plotFPCA(fpca_result)
```

`get_bone_connections` *Get bone connections from a skeleton*

Description

Returns the bone connections as an edge list (data.frame) or as an adjacency matrix.

Usage

```
get_bone_connections(skeleton, as_matrix = FALSE)
```

Arguments

skeleton	A SkeletonModel object.
as_matrix	Logical. If TRUE, return a square adjacency matrix with keypoint labels as row/column names. Default FALSE.

Value

If as_matrix = FALSE, a data.frame with columns from_label, to_label, and bone_name. If as_matrix = TRUE, a symmetric logical adjacency matrix.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

SkeletonModel(), define_skeleton(), get_segment_lengths()

Examples

```
sk <- define_skeleton("COCO")
edges <- get_bone_connections(sk)
head(edges)

adj <- get_bone_connections(sk, as_matrix = TRUE)
dim(adj)
```

---

get_limb_pairs *Get left/right limb pairs for symmetry analysis*

Description

Identifies corresponding left and right keypoint pairs in a skeleton model, useful for bilateral symmetry analysis.

Usage

```
get_limb_pairs(skeleton)
```

Arguments

skeleton	A SkeletonModel object.

Value

A `data.frame` with columns `left` and `right`, where each row is a matched pair of keypoint labels.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`SkeletonModel()`, `define_skeleton()`, `get_segment_lengths()`

Examples

```
sk <- define_skeleton("BODY_25")
pairs <- get_limb_pairs(sk)
head(pairs)
```

---

get_segment_lengths *Compute segment lengths from a PhysioExperiment and skeleton*

Description

Calculates the Euclidean distance between connected keypoints for each frame. The PhysioExperiment must contain `position_x`, `position_y`, and (optionally) `position_z` assays with columns matching skeleton keypoint labels.

Usage

```
get_segment_lengths(pe, skeleton)
```

Arguments

`pe`	A `PhysioExperiment` object with position assays.
`skeleton`	A `SkeletonModel` object whose keypoint labels match column names in the position assays.

Value

A matrix of segment lengths with dimensions (n_frames x n_bones). Column names are bone names.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`SkeletonModel()`, `get_bone_connections()`, `get_limb_pairs()`

Examples

```
## Not run:
pe <- readOpenPose("path/to/frames/", model = "BODY_25")
sk <- define_skeleton("BODY_25")
lengths <- get_segment_lengths(pe, sk)

## End(Not run)
```

`getEvent` *Get an event by name from a TaskSchema*

Description

Get an event by name from a TaskSchema

Usage

```
getEvent(schema, event_name)
```

Arguments

`schema`	A TaskSchema object
`event_name`	Name of the event to retrieve

Value

An Event object or NULL if not found

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`getEventNames()`, `getPhase()`, `TaskSchema()`

getEventNames *Get event names from a TaskSchema*

Description

Get event names from a TaskSchema

Usage

```
getEventNames(schema)
```

Arguments

schema	A TaskSchema object

Value

Character vector of event names

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

getPhaseNames(), getEvent(), TaskSchema()

getPhase *Get a phase by name from a TaskSchema*

Description

Get a phase by name from a TaskSchema

Usage

```
getPhase(schema, phase_name, search_subphases = TRUE)
```

Arguments

schema	A TaskSchema object
phase_name	Name of the phase to retrieve
search_subphases	Whether to search within subphases

Value

A Phase object or NULL if not found

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`getPhaseNames()`, `getPhaseColors()`, `getEvent()`

`getPhaseColors` *Get phase colors from a TaskSchema*

Description

Get phase colors from a TaskSchema

Usage

```
getPhaseColors(schema, include_subphases = FALSE)
```

Arguments

`schema`	A TaskSchema object
`include_subphases`	Whether to include subphase colors

Value

Named character vector of colors (phase name -> color)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`getPhaseNames()`, `getPhase()`, `TaskSchema()`

getPhaseData *Get data for all phases as a list of matrices*

Description

Get data for all phases as a list of matrices

Usage

```
getPhaseData(x, include_subphases = FALSE)
```

Arguments

x	A segmented_phases object
include_subphases	Whether to include subphases

Value

Named list of matrices

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

segmentPhases(), extractPhase(), hasValidPhases()

getPhaseNames *Get phase names from a TaskSchema*

Description

Get phase names from a TaskSchema

Usage

```
getPhaseNames(schema, include_subphases = FALSE)
```

Arguments

schema	A TaskSchema object
include_subphases	Whether to include subphase names

Value

Character vector of phase names

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`getEventNames()`, `getPhase()`, `getPhaseColors()`

`getSchema` *Get a pre-built schema by name*

Description

Get a pre-built schema by name

Usage

```
getSchema(name)
```

Arguments

`name`	Name of the schema ("gait", "running", "jump", "throw", "balance", "cutting", "cycling")

Value

A TaskSchema object

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`listSchemas()`, `TaskSchema()`, `validateSchema()`

Examples

```
gait <- getSchema("gait")
jump <- getSchema("jump")
```

hasValidPhases *Check if all phases are valid*

Description

Check if all phases are valid

Usage

```
hasValidPhases(x)
```

Arguments

x	A segmented_phases object

Value

Logical indicating if all phases have valid data

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

segmentPhases(), getPhaseData(), extractPhase()

icc *Intraclass Correlation Coefficient (ICC)*

Description

Computes the ICC using ANOVA-based methods following Shrout & Fleiss (1979). Supports one-way and two-way random/mixed models, agreement and consistency types, and single or average unit measures.

Usage

```
icc(
  ratings,
  model = c("twoway", "oneway"),
  type = c("agreement", "consistency"),
  unit = c("single", "average")
)
```

Arguments

`ratings`	Numeric matrix with subjects as rows and raters/sessions as columns.
`model`	Character; ICC model type: **"oneway"** One-way random effects (ICC(1,1) or ICC(1,k)) **"twoway"** Two-way random/mixed effects (default)
`type`	Character; agreement type: **"agreement"** Absolute agreement (ICC(2,1) for twoway) **"consistency"** Consistency/relative agreement (ICC(3,1) for twoway)
`unit`	Character; unit of measurement: **"single"** Reliability for a single rater/measurement **"average"** Reliability for the mean of k raters/measurements

Details

The function implements the six ICC forms from Shrout & Fleiss (1979):

- ICC(1,1): One-way random, single measures
- ICC(1,k): One-way random, average measures
- ICC(2,1): Two-way random, absolute agreement, single measures
- ICC(2,k): Two-way random, absolute agreement, average measures
- ICC(3,1): Two-way mixed, consistency, single measures
- ICC(3,k): Two-way mixed, consistency, average measures

Value

A list with components:

icc ICC value

ci_lower Lower bound of 95 percent confidence interval

ci_upper Upper bound of 95 percent confidence interval

f_value F statistic from ANOVA

p_value p-value for testing ICC = 0

model Model used

type Agreement type used

References

Shrout PE, Fleiss JL (1979). Intraclass correlations: Uses in assessing rater reliability. Psychological Bulletin, 86(2), 420-428.

See Also

`sem()` for standard error of measurement based on ICC, `mdc()` for minimal detectable change, `blandAltman()` for agreement analysis between methods.

Examples

```
# Test-retest reliability
ratings <- matrix(c(
  9, 2, 5, 8,
  6, 1, 3, 2,
  8, 4, 6, 8,
  7, 1, 2, 6,
  10, 5, 6, 9,
  6, 2, 4, 7
), nrow = 6, ncol = 4, byrow = TRUE)

result <- icc(ratings, model = "twoway", type = "agreement")
result$icc
```

---

integrateEMGMoCap *Integrate EMG and MoCap signals onto a common timeline*

Description

Aligns EMG to MoCap sample times and returns a combined table for downstream feature analysis.

Usage

```
integrateEMGMoCap(
  mocap,
  emg,
  mocap_sampling_rate = NULL,
  emg_sampling_rate,
  mocap_assay = NULL,
  process = TRUE,
  ...
)
```

Arguments

mocap	A numeric matrix/data.frame (time x features), or a PhysioExperiment object.
emg	Numeric vector or matrix (time x channels).
mocap_sampling_rate	MoCap sampling rate in Hz. If mocap is a PhysioExperiment and this is NULL, uses samplingRate(mocap).
emg_sampling_rate	EMG sampling rate in Hz.
mocap_assay	Assay name to use when mocap is a PhysioExperiment.
process	Logical; if TRUE, runs processEMG() before combining.
...	Additional arguments passed to processEMG().

Value

A list with mocap, emg_aligned, and combined data.frame.

References

Merletti R, Parker PA (2004). "Electromyography: Physiology, Engineering, and Non-Invasive Applications." IEEE Press/Wiley.

See Also

processEMG() for EMG processing pipeline, alignEMGtoMoCap() for time-alignment of EMG to MoCap, synchronizeSignals() for general multi-signal synchronization.

Examples

```
mocap <- matrix(rnorm(500), ncol = 5)
emg <- matrix(rnorm(5000), ncol = 2)
out <- integrateEMGMoCap(mocap, emg, mocap_sampling_rate = 100,
                         emg_sampling_rate = 1000)
```

---

inverseDynamics2D *Compute 2D lower-limb joint moments with inverse dynamics*

Description

Computes sagittal-plane ankle, knee, and hip net moments from joint-center coordinates, GRF, COP, and segment inertial terms.

Usage

```
inverseDynamics2D(
  joints,
  grf,
  sampling_rate,
  angles = NULL,
  angular_velocity = NULL,
  angular_acceleration = NULL,
  inertial = NULL,
  angle_unit = c("radian", "degree")
)
```

Arguments

joints	Matrix/data.frame with columns ankle_x, ankle_y, knee_x, knee_y, hip_x, hip_y.
grf	Matrix/data.frame with at least fx and fy columns, and optionally cop_x and cop_y.

sampling_rate	Sampling rate in Hz.
angles	Optional matrix/data.frame with columns ankle, knee, hip (joint angles).
angular_velocity	Optional matrix/data.frame with columns ankle, knee, hip.
angular_acceleration	Optional matrix/data.frame with columns ankle, knee, hip.
inertial	Optional data.frame from estimateSegmentInertia(). If omitted, inertial corrections are skipped.
angle_unit	Unit of angles: "radian" or "degree".

Value

A data.frame with time index and joint moments (ankle_moment, knee_moment, hip_moment). If angular velocity is available, corresponding power columns are included.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

inverseDynamics3D() for 3D inverse dynamics computation, estimateSegmentInertia() for segment inertial properties, computeJointPower() for joint power calculation.

Examples

```
n <- 200
joints <- data.frame(
  ankle_x = rep(0.00, n), ankle_y = rep(0.05, n),
  knee_x = rep(0.00, n),  knee_y = rep(0.45, n),
  hip_x = rep(0.00, n),   hip_y = rep(0.85, n)
)
grf <- data.frame(fx = rep(0, n), fy = abs(sin(seq(0, pi, length.out = n))) * 800,
                  cop_x = rep(0.02, n), cop_y = rep(0, n))
id <- inverseDynamics2D(joints, grf, sampling_rate = 100)
```

---

inverseDynamics3D *Compute 3D lower-limb joint moments with inverse dynamics*

Description

Computes ankle, knee, and hip net moments in 3D from joint-center coordinates, GRF, COP, and optional inertial terms.

Usage

```
inverseDynamics3D(
  joints,
  grf,
  sampling_rate,
  angles = NULL,
  angular_velocity = NULL,
  angular_acceleration = NULL,
  inertial = NULL,
  angle_unit = c("radian", "degree")
)
```

Arguments

joints	Matrix/data.frame with columns ankle_x, ankle_y, ankle_z, knee_x, knee_y, knee_z, hip_x, hip_y, hip_z.
grf	Matrix/data.frame with columns fx, fy, fz and optional cop_x, cop_y, cop_z.
sampling_rate	Sampling rate in Hz.
angles	Optional matrix/data.frame containing 3D joint angles with columns ankle_x, ankle_y, ankle_z, knee_x, knee_y, knee_z, hip_x, hip_y, hip_z.
angular_velocity	Optional 3D joint angular velocity table.
angular_acceleration	Optional 3D joint angular acceleration table.
inertial	Optional data.frame from estimateSegmentInertia(). If supplied with angular acceleration, inertial moment terms are added.
angle_unit	Unit of angle-related inputs: "radian" or "degree".

Value

A data.frame with time and moment components: *_moment_x, *_moment_y, *_moment_z. If angular velocity is available, *_power_total columns are included.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

inverseDynamics2D() for sagittal-plane inverse dynamics, estimateSegmentInertia() for segment inertial properties, computeJointPower() for joint power calculation.

Examples

```
n <- 200
joints <- data.frame(
  ankle_x = rep(0.00, n), ankle_y = rep(0.00, n), ankle_z = rep(0.05, n),
  knee_x = rep(0.00, n),  knee_y = rep(0.00, n),  knee_z = rep(0.45, n),
  hip_x = rep(0.00, n),   hip_y = rep(0.00, n),   hip_z = rep(0.85, n)
)
grf <- data.frame(
  fx = rep(50, n), fy = rep(0, n), fz = rep(700, n),
  cop_x = rep(0.02, n), cop_y = rep(0, n), cop_z = rep(0, n)
)
out <- inverseDynamics3D(joints, grf, sampling_rate = 100)
```

---

listSchemas *Get list of all pre-built schemas*

Description

Get list of all pre-built schemas

Usage

```
listSchemas()
```

Value

Named list of all available pre-built TaskSchema objects

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

getSchema(), TaskSchema(), validateSchema()

Examples

```
schemas <- listSchemas()
names(schemas)
```

`manualEvents` *Manually specify event times*

Description

Create a detected_events object from manually specified times or indices.

Usage

```
manualEvents(
  schema,
  times = NULL,
  indices = NULL,
  sampling_rate = NULL,
  n_samples
)
```

Arguments

`schema`	TaskSchema object
`times`	Named numeric vector of event times in seconds
`indices`	Named integer vector of event indices
`sampling_rate`	Sampling rate (required if using times)
`n_samples`	Total number of samples (required)

Value

A detected_events data.frame

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`detectEvents()`, `segmentPhases()`, `TaskSchema()`

Examples

```
events <- manualEvents(
  schema_gait,
  times = c(hs1 = 0, to = 0.6, hs2 = 1.0),
  sampling_rate = 1000,
  n_samples = 1000
)
```

mdc *Minimal Detectable Change (MDC)*

Description

Computes the minimal detectable change from the standard error of measurement.

Usage

```
mdc(sem_value, confidence = 0.95)
```

Arguments

sem_value	Numeric; the standard error of measurement.
confidence	Numeric; confidence level (default 0.95).

Details

MDC is computed as:

$$MDC = SEM \times z \times \sqrt{2}$$

where $z = \Phi^{-1}(1 - (1 - confidence)/2)$.

At 95\

Value

Numeric MDC value.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

See Also

sem() for computing standard error of measurement, icc() for computing intraclass correlation coefficients.

Examples

```
sem_val <- 2.5
mdc(sem_val, confidence = 0.95)
```

`movingAverage` *Simple moving average filter*

Description

Applies a symmetric moving average for quick smoothing of signal data. Uses `stats::filter()` internally with equal weights.

Usage

```
movingAverage(x, window = 5)
```

Arguments

`x`	A numeric vector or matrix (time x channels).
`window`	Window size for the moving average (default: 5). Will be coerced to an odd integer.

Value

Smoothed data with the same dimensions as `x`. Edge values where the full window cannot be applied are set to NA.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`butterworthFilter()` for Butterworth low-pass filtering, `savgolFilter()` for Savitzky-Golay polynomial smoothing.

Examples

```
# Smooth a noisy signal
x <- sin(seq(0, 4 * pi, length.out = 200)) + rnorm(200, sd = 0.3)
x_smooth <- movingAverage(x, window = 7)
```

`normalizedTimeAxis` *Time axis for normalized data*

Description

Generates an appropriate time axis for normalized data.

Usage

```
normalizedTimeAxis(
  norm_length,
  method = "cycle",
  unit = c("percent", "normalized", "degrees")
)
```

Arguments

`norm_length`	Length of normalized data
`method`	Normalization method used
`unit`	Output unit ("percent", "normalized", "degrees")

Value

Numeric vector of time values

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`normalizeMovement()`, `batchNormalize()`, `segmentPhases()`

Examples

```
# 0-100% axis
t_pct <- normalizedTimeAxis(101, method = "cycle", unit = "percent")

# 0-360 degrees for cycling
t_deg <- normalizedTimeAxis(361, method = "cycle", unit = "degrees")
```

`normalizeEMG` *Normalize EMG by MVC or peak*

Description

Normalize EMG by MVC or peak

Usage

```
normalizeEMG(x, method = c("mvc", "peak"), mvc = NULL, scale_percent = TRUE)
```

Arguments

`x`	Numeric vector or matrix (time x channels).
`method`	Normalization method: `"mvc"` or `"peak"`.
`mvc`	Numeric scalar or vector of MVC reference values. Required when `method = "mvc"`.
`scale_percent`	Logical; if `TRUE`, returns percent scale (x100).

Value

Normalized signal with same dimensions as `x`.

References

Merletti R, Parker PA (2004). "Electromyography: Physiology, Engineering, and Non-Invasive Applications." IEEE Press/Wiley.

See Also

`rectifyEMG()` for signal rectification, `computeRMSEnvelope()` for RMS envelope computation, `processEMG()` for complete EMG processing pipeline.

Examples

```
x <- matrix(abs(rnorm(500)), ncol = 2)
normalizeEMG(x, method = "peak")
```

`normalizeMovement` *Normalize movement data*

Description

Normalizes movement data using various methods appropriate for different task types.

Usage

```
normalizeMovement(
  x,
  method = c("cycle", "phase", "landmark", "dtw", "absolute"),
  norm_length = 101L,
  schema = NULL,
  events = NULL,
  reference = NULL,
  ...
)
```

Arguments

`x`	Data to normalize: • PhysioExperiment object • Matrix (time x channels) • segmented_phases object • List of matrices (multiple trials)
`method`	Normalization method: • "cycle" - Normalize entire movement to fixed length (0-100%) • "phase" - Normalize each phase independently • "landmark" - Align to a specific event • "dtw" - Dynamic time warping alignment • "absolute" - Keep original time (no normalization)
`norm_length`	Target length after normalization (default 101 for 0-100%)
`schema`	Optional TaskSchema for context
`events`	Optional detected_events for landmark alignment
`reference`	Reference trial for DTW (default: mean of all trials)
`...`	Additional arguments passed to specific methods

Value

Normalized data in the same format as input (or matrix for segmented_phases)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

batchNormalize(), normalizedTimeAxis(), segmentPhases()

Examples

```
# Normalize a matrix to 101 points
data <- matrix(rnorm(500), nrow = 500, ncol = 3)
normalized <- normalizeMovement(data, method = "cycle", norm_length = 101)
```

Phase *Create a Phase definition*

Description

Defines a phase within a movement task, bounded by start and end events.

Usage

```
Phase(name, label, start_event, end_event, color = NULL, subphases = list())
```

Arguments

name	Short identifier for the phase (e.g., "stance", "swing")
label	Human-readable label (e.g., "Stance Phase")
start_event	Name of the event marking phase start
end_event	Name of the event marking phase end
color	Optional color for visualization (hex code)
subphases	Optional list of Phase objects for nested phases

Value

A Phase object (S3 class)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

Event(), TaskSchema(), segmentPhases()

Examples

```
# Simple phase definition
stance <- Phase("stance", "Stance Phase", "hs1", "to", color = "#E8F4F8")

# Phase with subphases
stance <- Phase("stance", "Stance Phase", "hs1", "to", color = "#E8F4F8",
                subphases = list(
                  Phase("loading", "Loading Response", "hs1", "ff"),
                  Phase("midstance", "Midstance", "ff", "ho"),
                  Phase("propulsion", "Propulsion", "ho", "to")
                ))
```

---

phaseDurations *Get phase durations*

Description

Get phase durations

Usage

```
phaseDurations(x, unit = c("percent", "seconds", "samples"))
```

Arguments

x	A segmented_phases object
unit	"samples", "seconds", or "percent"

Value

Named numeric vector of phase durations

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

phaseTiming(), phaseRatios(), segmentPhases()

phaseRatios *Calculate phase ratios*

Description

Calculate phase ratios

Usage

```
phaseRatios(x, reference = "total")
```

Arguments

x	A segmented_phases object
reference	Reference for ratio calculation ("total" or phase name)

Value

Named numeric vector of phase ratios

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

phaseDurations(), phaseTiming(), segmentPhases()

phaseTiming *Get phase timing information*

Description

Get phase timing information

Usage

```
phaseTiming(x, as_percent = TRUE)
```

Arguments

x	A segmented_phases object
as_percent	Return timing as percentage (TRUE) or seconds (FALSE)

Value

A data.frame with phase timing information

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

segmentPhases(), phaseDurations(), phaseRatios()

---

plotCorrelationMatrix *Plot correlation matrix heatmap*

Description

Creates a heatmap visualization of a correlation matrix with optional clustering and significance masking.

Usage

```
plotCorrelationMatrix(
  x,
  method = "pearson",
  cluster = TRUE,
  show_values = TRUE,
  show_significance = FALSE,
  alpha = 0.05,
  colors = c("#B2182B", "white", "#2166AC"),
  title = "Correlation Matrix"
)
```

Arguments

x	A correlation matrix, data.frame, or PhysioExperiment.
method	If x is data, correlation method: "pearson", "spearman", "kendall".
cluster	Logical; apply hierarchical clustering to reorder.
show_values	Logical; display correlation values in cells.
show_significance	Logical; mask non-significant correlations.
alpha	Significance threshold for masking.
colors	Color palette (low, mid, high).
title	Plot title.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

`plotEffectSizeForest()` for forest plots of effect sizes, `plotWaveformComparison()` for comparing waveform patterns across groups.

Examples

```
# Correlation matrix of joint angles
set.seed(123)
data <- data.frame(
  Hip = rnorm(100),
  Knee = rnorm(100),
  Ankle = rnorm(100)
)
data$Knee <- data$Hip * 0.7 + rnorm(100, 0, 0.5)  # Correlated

plotCorrelationMatrix(data)
```

---

plotCycle *Plot normalized movement cycle*

Description

Creates a waveform plot with automatic phase and event annotations based on the TaskSchema. This is a generalized version of plotGaitCycle.

Usage

```
plotCycle(
  x,
  schema = NULL,
  events = NULL,
  channel = 1L,
  show_mean = TRUE,
  show_sd = TRUE,
  show_ci = FALSE,
  ci = 0.95,
  show_individual = FALSE,
  show_events = NULL,
  show_phases = NULL,
  time_axis = NULL,
```

```
  xlab = NULL,
  ylab = "Value",
  title = NULL,
  colors = NULL,
  ...
)
```

Arguments

x	Normalized data (matrix, PhysioExperiment, or 3D array)
schema	TaskSchema object for formatting and annotations
events	Optional detected_events for event markers
channel	Channel index or name to plot (for multi-channel data)
show_mean	Show mean line
show_sd	Show standard deviation band
show_ci	Show confidence interval band
ci	Confidence level (default 0.95)
show_individual	Show individual trials
show_events	Show event markers
show_phases	Show phase regions
time_axis	Custom time axis values
xlab	X-axis label (default from schema)
ylab	Y-axis label
title	Plot title
colors	Named vector of colors for phases
...	Additional arguments passed to ggplot

Value

A ggplot object

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

`plotGroupComparison()` for multi-group comparisons, `plotMultiPanel()` for multi-channel cycle visualization, `plotPhaseDurations()` for phase duration bar charts.

Examples

```
# Basic usage with schema
data <- matrix(rnorm(101 * 10), nrow = 101)
p <- plotCycle(data, schema = schema_gait)

# With events
events <- manualEvents(schema_gait, c(hs1 = 0, to = 0.6, hs2 = 1.0),
                       sampling_rate = 100, n_samples = 101)
p <- plotCycle(data, schema = schema_gait, events = events)
```

---

plotDTW *Plot DTW alignment*

Description

Visualizes the DTW alignment between two waveforms.

Usage

```
plotDTW(
  dtw_result,
  x = NULL,
  y = NULL,
  type = c("alignment", "cost", "waveforms", "all")
)
```

Arguments

dtw_result	A dtw_result object from dtwDistance().
x	First waveform (optional, for overlay).
y	Second waveform (optional, for overlay).
type	Plot type: "alignment", "cost", "waveforms", or "all".

Value

A ggplot object or list of plots.

References

Sakoe H, Chiba S (1978). "Dynamic programming algorithm optimization for spoken word recognition." IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49.

See Also

dtwDistance(), dtwWarp(), dtwClustering()

`plotEffectSizeForest` *Plot effect size forest plot*

Description

Creates a forest plot displaying effect sizes with confidence intervals, commonly used for meta-analysis style visualization of multiple comparisons.

Usage

```
plotEffectSizeForest(
  effects,
  ci_lower,
  ci_upper,
  labels = NULL,
  null_value = 0,
  sort_by = c("none", "effect", "name"),
  title = "Effect Sizes"
)
```

Arguments

`effects`	Named vector or data.frame of effect sizes.
`ci_lower`	Lower confidence interval bounds.
`ci_upper`	Upper confidence interval bounds.
`labels`	Labels for each effect (uses names if not provided).
`null_value`	Reference line value (default: 0).
`sort_by`	How to sort: "none", "effect", "name".
`title`	Plot title.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

`cohensD()` for computing Cohen's d effect sizes, `etaSquared()` for computing eta-squared effect sizes, `plotCorrelationMatrix()` for correlation heatmaps.

Examples

```
# Forest plot of effect sizes across joints
effects <- c(Hip = 0.8, Knee = 1.2, Ankle = 0.3)
ci_lower <- c(0.4, 0.8, -0.1)
ci_upper <- c(1.2, 1.6, 0.7)

plotEffectSizeForest(effects, ci_lower, ci_upper)
```

---

plotFPCA *Plot fPCA results*

Description

Visualizes functional PCA results including loadings and variance explained.

Usage

```
plotFPCA(
  x,
  type = c("loadings", "variance", "scores", "all"),
  components = 1:4,
  time_axis = NULL
)
```

Arguments

x	An fpca_result object.
type	Plot type: "loadings", "variance", "scores", or "all".
components	Which components to plot.
time_axis	Optional time axis values.

Value

A ggplot object (or list of plots for type = "all").

References

Ramsay JO, Silverman BW (2005). "Functional Data Analysis." 2nd ed. Springer.

See Also

fPCA() for computing fPCA results, reconstructFPCA() for waveform reconstruction.

`plotGaitCycle` *Plot gait cycle normalized waveforms*

Description

Plots waveforms normalized to gait cycle (0-100%) with options for displaying multiple trials, mean, and variability bands.

Usage

```
plotGaitCycle(
  x,
  events = NULL,
  normalize_to = 101,
  show_events = TRUE,
  show_mean = TRUE,
  show_sd = TRUE,
  show_individual = TRUE,
  event_labels = c(HS = 0, TO = 60, HS = 100),
  title = "Gait Cycle",
  ylab = "Value"
)
```

Arguments

`x`	A PhysioExperiment, matrix, or list of matrices.
`events`	Optional data.frame with gait events (heel strike, toe off).
`normalize_to`	Length to normalize (default: 101 for 0-100%).
`show_events`	Logical; show vertical lines at gait events.
`show_mean`	Logical; show mean waveform.
`show_sd`	Logical; show SD bands.
`show_individual`	Logical; show individual trials.
`event_labels`	Labels for gait events.
`title`	Plot title.
`ylab`	Y-axis label.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

plotWaveformComparison() for multi-group comparisons, plotSpaghetti() for individual waveform overlays, calculateGaitParameters() for computing gait metrics.

Examples

```
# Normalize and plot knee angle across gait cycle
set.seed(123)
data <- sapply(1:10, function(i) {
  n <- sample(90:110, 1)  # Variable cycle length
  sin(seq(0, 2*pi, length.out = n)) * 60 + rnorm(n, 0, 3)
})

plotGaitCycle(data, show_mean = TRUE, show_sd = TRUE,
              ylab = "Knee Flexion (deg)")
```

---

plotGroupComparison *Plot group comparison with task context*

Description

Compares waveforms between groups with schema-aware formatting.

Usage

```
plotGroupComparison(
  x,
  groups,
  schema = NULL,
  events = NULL,
  show_individual = FALSE,
  ci = 0.95,
  xlab = NULL,
  ylab = "Value",
  title = NULL,
  group_colors = NULL,
  ...
)
```

Arguments

x	Normalized data (matrix: time x samples)
groups	Factor or character vector indicating group membership
schema	TaskSchema object
events	Optional detected_events
show_individual	Show individual waveforms

ci	Confidence interval level
xlab	X-axis label
ylab	Y-axis label
title	Plot title
group_colors	Named vector of colors for groups
...	Additional arguments

Value

A ggplot object

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

plotCycle() for single-group cycle visualization, plotWaveformComparison() for alternative group comparison plots.

---

plotMultiPanel *Plot multi-panel movement data*

Description

Creates a faceted plot showing multiple channels or variables.

Usage

```
plotMultiPanel(
  x,
  schema = NULL,
  channels = NULL,
  facet_scales = "free_y",
  show_mean = TRUE,
  show_sd = TRUE,
  ...
)
```

Arguments

x	Normalized data (matrix or 3D array)
schema	TaskSchema object
channels	Channels to plot (indices or names)
facet_scales	Scales for faceting ("free_y", "fixed", etc.)
show_mean	Show mean across trials
show_sd	Show SD bands
...	Additional arguments passed to plotCycle

Value

A ggplot object

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

`plotCycle()` for single-channel cycle visualization, `plotGroupComparison()` for between-group comparisons.

`plotPCAScatter` *Plot PCA scatter*

Description

Creates a scatter plot of PCA scores.

Usage

```
plotPCAScatter(x, components = c(1, 2), groups = NULL, labels = NULL)
```

Arguments

`x`	A waveform_pca object.
`components`	Which PCs to plot (length 2).
`groups`	Optional grouping factor for coloring.
`labels`	Optional labels for points.

Value

A ggplot object.

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

`waveformPCA()`, `plotPCAVariance()`, `plotUMAP()`

plotPCAVariance *Plot PCA variance explained*

Description

Creates a scree plot showing variance explained by each PC.

Usage

```
plotPCAVariance(x, n_components = NULL)
```

Arguments

x	A waveform_pca object.
n_components	Number of components to show.

Value

A ggplot object.

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

waveformPCA(), plotPCAScatter(), plotUMAP()

plotPhaseDurations *Plot phase duration comparison*

Description

Visualizes phase durations across conditions or groups.

Usage

```
plotPhaseDurations(
  phases_list,
  labels = NULL,
  unit = c("percent", "seconds"),
  show_values = TRUE,
  title = "Phase Durations",
  ...
)
```

Arguments

phases_list	List of segmented_phases objects or timing data.frames
labels	Labels for each entry
unit	"percent" or "seconds"
show_values	Show duration values on bars
title	Plot title
...	Additional arguments

Value

A ggplot object

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

plotCycle() for cycle-normalized waveform visualization, calculateGaitParameters() for computing gait phase parameters.

---

plotPhasePortrait *Plot phase portrait*

Description

Creates a phase portrait (angle vs angular velocity) for analyzing movement dynamics and coordination patterns.

Usage

```
plotPhasePortrait(
  angle,
  velocity = NULL,
  sampling_rate = 100,
  groups = NULL,
  normalize = FALSE,
  title = "Phase Portrait"
)
```

Arguments

angle	Angle time series (vector or matrix).
velocity	Angular velocity. If NULL, computed from angle.
sampling_rate	Sampling rate (needed if velocity computed).
groups	Optional grouping factor.
normalize	Logical; normalize to unit circle.
title	Plot title.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

computeVelocity() for computing angular velocities from position data, plotGaitCycle() for time-domain gait cycle visualization.

Examples

```
# Phase portrait of knee angle
set.seed(123)
t <- seq(0, 2*pi, length.out = 100)
angle <- sin(t) * 60
velocity <- cos(t) * 60 * (2*pi/100)  # Derivative

plotPhasePortrait(angle, velocity)
```

---

plotSkeleton *Plot a 2D stick-figure skeleton*

Description

Draws a single frame from a PhysioExperiment as a 2D stick figure by projecting 3D marker positions onto an anatomical plane.

Usage

```
plotSkeleton(
  pe,
  skeleton,
  frame = 1,
  plane = c("sagittal", "frontal", "transverse"),
  show_labels = FALSE,
  show_confidence = FALSE,
  point_size = 3,
  segment_color = "gray40",
  title = NULL
)
```

Arguments

pe	A PhysioExperiment object with position_x, position_y, and position_z assays (columns named by keypoint label).
skeleton	A SkeletonModel object whose keypoint labels match column names in the position assays.
frame	Integer frame (row) index to plot (default 1).
plane	Anatomical plane for projection: "sagittal" (drop X), "frontal" (drop Y), or "transverse" (drop Z).
show_labels	Logical; annotate markers with their labels.
show_confidence	Logical; color markers by confidence if a confidence assay is present.
point_size	Marker point size (default 3).
segment_color	Color for bone segments (default "gray40").
title	Plot title. If NULL, auto-generated.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

plotSkeletonSequence() for multi-frame faceted display, plotSkeletonOverlay() for overlaid motion visualization, plotSkeleton3D() for pseudo-3D rendering, projectTo2D() for coordinate projection utilities.

Examples

```
## Not run:
pe <- readOpenPose("frames/", model = "BODY_25")
sk <- define_skeleton("BODY_25")
plotSkeleton(pe, sk, frame = 1, plane = "frontal")

## End(Not run)
```

`plotSkeleton3D` *Plot a skeleton frame in pseudo-3D*

Description

Renders one frame of a skeleton in a perspective-like 3D projection using a lightweight base graphics backend (no external 3D dependency required).

Usage

```
plotSkeleton3D(
  pe,
  skeleton,
  frame = 1,
  azimuth = 35,
  elevation = 20,
  distance = 6,
  show_labels = FALSE,
  point_col = "steelblue",
  segment_col = "gray40",
  point_cex = 1.1,
  segment_lwd = 1.5,
  main = NULL,
  draw = TRUE
)
```

Arguments

`pe`	A PhysioExperiment object with `position_x`, `position_y`, and `position_z` assays.
`skeleton`	A `SkeletonModel` object.
`frame`	Integer frame index.
`azimuth`	View azimuth angle in degrees.
`elevation`	View elevation angle in degrees.
`distance`	Perspective distance (larger = weaker perspective).
`show_labels`	Logical; if `TRUE`, draws keypoint labels.
`point_col`	Point color.

segment_col	Segment color.
point_cex	Point size.
segment_lwd	Segment line width.
main	Plot title. If NULL, auto-generated.
draw	Logical; if TRUE, draws to current graphics device.

Value

Invisibly returns a list with projected coordinates and view parameters.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

plotSkeleton() for 2D skeleton visualization, projectTo2D() for manual coordinate projection.

Examples

```
## Not run:
pe <- readOpenPose("frames/", model = "BODY_25")
sk <- define_skeleton("BODY_25")
plotSkeleton3D(pe, sk, frame = 10, azimuth = 35, elevation = 20)

## End(Not run)
```

---

plotSkeletonOverlay *Plot overlaid skeleton frames*

Description

Draws multiple frames on the same plot. Earlier frames are rendered with lower opacity when alpha_decay = TRUE.

Usage

```
plotSkeletonOverlay(
  pe,
  skeleton,
  frames,
  plane = c("sagittal", "frontal", "transverse"),
  alpha_decay = TRUE,
  base_alpha = 0.2,
  show_labels = FALSE,
  point_size = 3,
```

```
  segment_color = "gray40",
  title = NULL
)
```

Arguments

pe	A PhysioExperiment object with position_x, position_y, and position_z assays (columns named by keypoint label).
skeleton	A SkeletonModel object whose keypoint labels match column names in the position assays.
frames	Integer vector of frame indices to overlay.
plane	Anatomical plane for projection: "sagittal" (drop X), "frontal" (drop Y), or "transverse" (drop Z).
alpha_decay	Logical; if TRUE, earlier frames have lower opacity (linearly from 0.2 to 1.0).
base_alpha	Minimum alpha for the earliest frame (default 0.2).
show_labels	Logical; annotate markers with their labels.
point_size	Marker point size (default 3).
segment_color	Color for bone segments (default "gray40").
title	Plot title. If NULL, auto-generated.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

plotSkeleton() for single-frame skeleton visualization, plotSkeletonSequence() for multi-frame faceted display.

Examples

```
## Not run:
pe <- readOpenPose("frames/", model = "BODY_25")
sk <- define_skeleton("BODY_25")
plotSkeletonOverlay(pe, sk, frames = c(1, 5, 10, 15, 20))

## End(Not run)
```

`plotSkeletonSequence` *Plot a sequence of skeleton frames*

Description

Creates a multi-panel faceted plot showing the skeleton at several frames.

Usage

```
plotSkeletonSequence(
  pe,
  skeleton,
  frames,
  plane = c("sagittal", "frontal", "transverse"),
  ncol = 3,
  show_labels = FALSE,
  show_confidence = FALSE,
  point_size = 2,
  segment_color = "gray40",
  title = NULL
)
```

Arguments

`pe`	A PhysioExperiment object with `position_x`, `position_y`, and `position_z` assays (columns named by keypoint label).
`skeleton`	A `SkeletonModel` object whose keypoint labels match column names in the position assays.
`frames`	Integer vector of frame indices to plot.
`plane`	Anatomical plane for projection: `"sagittal"` (drop X), `"frontal"` (drop Y), or `"transverse"` (drop Z).
`ncol`	Number of columns in the faceted layout (default 3).
`show_labels`	Logical; annotate markers with their labels.
`show_confidence`	Logical; color markers by confidence if a `confidence` assay is present.
`point_size`	Marker point size (default 3).
`segment_color`	Color for bone segments (default `"gray40"`).
`title`	Plot title. If `NULL`, auto-generated.

Value

A ggplot object with facets.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

plotSkeleton() for single-frame skeleton visualization, plotSkeletonOverlay() for overlaid frames on a single plot.

Examples

```
## Not run:
pe <- readOpenPose("frames/", model = "BODY_25")
sk <- define_skeleton("BODY_25")
plotSkeletonSequence(pe, sk, frames = c(1, 10, 20), ncol = 3)

## End(Not run)
```

---

plotSpaghetti *Plot spaghetti plot (all subjects with mean)*

Description

Creates a spaghetti plot showing individual subject waveforms with a highlighted mean trajectory.

Usage

```
plotSpaghetti(
  x,
  time_axis = NULL,
  highlight_mean = TRUE,
  individual_color = "gray60",
  mean_color = "red",
  alpha = 0.4,
  title = NULL,
  xlab = "Time",
  ylab = "Value"
)
```

Arguments

x	A PhysioExperiment or matrix (time x observations).
time_axis	Optional time axis values.
highlight_mean	Logical; highlight mean with thick line.
individual_color	Color for individual lines.
mean_color	Color for mean line.
alpha	Transparency for individual lines.
title	Plot title.
xlab	X-axis label.
ylab	Y-axis label.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

`plotGaitCycle()` for gait cycle visualization with event markers, `plotWaveformComparison()` for multi-group waveform comparisons.

`plotSymmetry` *Plot symmetry comparison (left vs right)*

Description

Creates a symmetry plot comparing bilateral waveforms, useful for gait symmetry analysis.

Usage

```
plotSymmetry(
  left,
  right,
  time_axis = NULL,
  ci = 0.95,
  show_diagonal = TRUE,
  plot_type = c("overlay", "scatter"),
  title = "Symmetry Analysis"
)
```

Arguments

`left`	Matrix of left side waveforms (time x observations).
`right`	Matrix of right side waveforms (time x observations).
`time_axis`	Optional time axis values.
`ci`	Confidence interval level.
`show_diagonal`	Logical; show y=x diagonal reference line.
`plot_type`	"overlay" for time series, "scatter" for L vs R scatter.
`title`	Plot title.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

`plotWaveformComparison()` for multi-group waveform comparisons, `calculateStepSymmetry()` for quantifying gait symmetry, `symmetryIndex()` for computing symmetry indices.

Examples

```
# Compare left and right knee angles
set.seed(123)
t <- seq(0, 100, length.out = 101)
left <- sapply(1:20, function(i) sin(2*pi*t/100) * 30 + rnorm(101, 0, 2))
right <- sapply(1:20, function(i) sin(2*pi*t/100) * 28 + rnorm(101, 0, 2))  # Slight asymmetry

plotSymmetry(left, right, time_axis = t, plot_type = "overlay")
```

---

plotTrajectory *Plot movement trajectory (2D)*

Description

Plots 2D trajectory, useful for balance (CoP) or cutting (CoM) analysis.

Usage

```
plotTrajectory(
  x,
  y,
  schema = NULL,
  show_path = TRUE,
  show_points = FALSE,
  show_ellipse = TRUE,
  ellipse_ci = 0.95,
  show_start_end = TRUE,
  color_by = NULL,
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  ...
)
```

Arguments

x	X-coordinate data
y	Y-coordinate data
schema	TaskSchema object
show_path	Show trajectory path
show_points	Show individual points
show_ellipse	Show confidence ellipse
ellipse_ci	Confidence level for ellipse
show_start_end	Mark start and end points
color_by	Optional variable for coloring (e.g., time)
xlab	X-axis label
ylab	Y-axis label
title	Plot title
...	Additional arguments

Value

A ggplot object

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

plotSkeleton() for full-body skeleton visualization, plotPhasePortrait() for phase-space trajectory plots.

---

plotUMAP *Plot UMAP embedding*

Description

Creates a scatter plot of UMAP embedding.

Usage

```
plotUMAP(x, groups = NULL, labels = NULL)
```

Arguments

x	A waveform_umap object.
groups	Optional grouping factor.
labels	Optional point labels.

Value

A ggplot object.

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

`waveformUMAP()`, `plotPCAScatter()`, `waveformPCA()`

`plotWaveformComparison`

Biomechanics-specific Visualization Functions

Description

Visualization functions designed for biomechanical waveform analysis including group comparisons, gait cycle plots, and symmetry analysis. Plot waveform comparison across groups

Usage

```
plotWaveformComparison(
  x,
  groups,
  channel = 1L,
  ci = 0.95,
  show_individual = FALSE,
  time_axis = NULL,
  colors = NULL,
  title = NULL,
  xlab = "Time",
  ylab = "Value"
)
```

Arguments

`x`	A PhysioExperiment object or matrix (time x observations).
`groups`	Factor or vector indicating group membership.
`channel`	For PhysioExperiment with multiple channels, which to plot.
`ci`	Confidence interval level (default: 0.95).
`show_individual`	Logical; show individual waveforms as thin lines.
`time_axis`	Optional time axis values (e.g., 0-100 for gait cycle).

colors	Optional color palette for groups.
title	Plot title.
xlab	X-axis label.
ylab	Y-axis label.

Details

Creates a comparison plot showing mean waveforms with confidence bands for multiple groups. Ideal for comparing gait patterns between conditions.

Value

A ggplot object.

References

Wickham H (2016). "ggplot2: Elegant Graphics for Data Analysis." Springer.

See Also

plotGaitCycle() for single-group gait cycle visualization, plotSymmetry() for left-right symmetry plots, plotSpaghetti() for individual waveform overlay plots.

Examples

```
# Compare gait patterns between groups
set.seed(123)
# Control group
control <- sapply(1:15, function(i) sin(seq(0, 2*pi, length.out = 101)) * 30 + rnorm(101, 0, 3))
# Patient group (reduced range of motion)
patient <- sapply(1:15, function(i) sin(seq(0, 2*pi, length.out = 101)) * 20 + rnorm(101, 0, 3))

data <- cbind(control, patient)
groups <- factor(rep(c("Control", "Patient"), each = 15))

plotWaveformComparison(data, groups, time_axis = 0:100,
                       xlab = "Gait Cycle (%)", ylab = "Knee Angle (deg)")
```

---

print.ASFSkeleton *Print method for ASFSkeleton objects*

Description

Print method for ASFSkeleton objects

Usage

```
## S3 method for class 'ASFSkeleton'
print(x, ...)
```

Arguments

x	An ASFSkeleton object.
...	Additional arguments (ignored).

Value

Invisibly returns x.

References

CMU Graphics Lab (2003). "CMU Motion Capture Database." http://mocap.cs.cmu.edu/.

See Also

readASF() for reading ASF skeleton files, readAMC() for reading AMC motion data.

---

print.benchmark_agreement

Print benchmark agreement summary

Description

Print benchmark agreement summary

Usage

```
## S3 method for class 'benchmark_agreement'
print(x, ...)
```

Arguments

x	A benchmark_agreement object.
...	Additional arguments (unused).

Value

Invisibly returns x.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

See Also

benchmarkAgreement() for computing agreement metrics.

print.benchmark_manifest_validation *Print benchmark manifest validation result*

Description

Print benchmark manifest validation result

Usage

```
## S3 method for class 'benchmark_manifest_validation'
print(x, ...)
```

Arguments

x	A benchmark_manifest_validation object.
...	Additional arguments (unused).

Value

Invisibly returns x.

References

Bland JM, Altman DG (1986). "Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement." Lancet, 327(8476), 307-310.

See Also

validateBenchmarkManifest() for performing manifest validation.

print.benchmark_suite *Print benchmark suite summary*

Description

Print benchmark suite summary

Usage

```
## S3 method for class 'benchmark_suite'
print(x, ...)
```

Arguments

x	A benchmark_suite object.
...	Additional arguments (unused).

Value

Invisibly returns `x`.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

See Also

`runBenchmarkSuite()` for running the benchmark suite.

`print.detected_events` *Print method for detected events*

Description

Print method for detected events

Usage

```
## S3 method for class 'detected_events'
print(x, ...)
```

Arguments

`x`	A detected_events object
`...`	Additional arguments (unused)

`print.dtw_clustering` *Print DTW clustering result*

Description

Print DTW clustering result

Usage

```
## S3 method for class 'dtw_clustering'
print(x, ...)
```

Arguments

`x`	A dtw_clustering object
`...`	Additional arguments (unused)

print.dtw_result *Print DTW result*

Description

Print DTW result

Usage

```
## S3 method for class 'dtw_result'
print(x, ...)
```

Arguments

x	A dtw_result object
...	Additional arguments (unused)

print.forceplate_analysis *Print a forceplate analysis summary*

Description

Print a forceplate analysis summary

Usage

```
## S3 method for class 'forceplate_analysis'
print(x, ...)
```

Arguments

x	A `forceplate_analysis` object.
...	Additional arguments (ignored).

Value

Invisibly returns `x`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`analyzeForcePlate()` for performing force plate analysis, `analyzeForcePlatePE()` for PhysioExperiment-based analysis.

print.forceplate_analysis_multi *Print a multi-plate forceplate analysis summary*

Description

Print a multi-plate forceplate analysis summary

Usage

```
## S3 method for class 'forceplate_analysis_multi'
print(x, ...)
```

Arguments

x	A forceplate_analysis_multi object.
...	Additional arguments (ignored).

Value

Invisibly returns x.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

analyzeForcePlatePE() for multi-plate force plate analysis, print.forceplate_analysis() for single-plate result display.

print.fpca_result *Print fPCA result*

Description

Print fPCA result

Usage

```
## S3 method for class 'fpca_result'
print(x, ...)
```

Arguments

x	An fpca_result object
...	Additional arguments (unused)

References

Ramsay JO, Silverman BW (2005). "Functional Data Analysis." 2nd ed. Springer.

See Also

`fPCA()` for performing functional principal component analysis, `plotFPCA()` for visualization of fPCA results, `reconstructFPCA()` for waveform reconstruction from fPCA scores.

`print.gait_parameters` *Print gait parameters*

Description

S3 print method showing mean +/- SD for each parameter.

Usage

```
## S3 method for class 'gait_parameters'
print(x, ...)
```

Arguments

x	A `gait_parameters` object.
...	Additional arguments (unused).

References

Perry J, Burnfield JM (2010). "Gait Analysis: Normal and Pathological Function." 2nd ed. SLACK Incorporated.

See Also

`calculateGaitParameters()` for computing gait parameters, `summarizeGaitParameters()` for descriptive statistics.

print.mocap_quickstart *Print a quick-start summary*

Description

Print a quick-start summary

Usage

```
## S3 method for class 'mocap_quickstart'
print(x, ...)
```

Arguments

x	A mocap_quickstart object.
...	Additional arguments (unused).

Value

Invisibly returns x.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

quickStartMoCap() for the complete getting-started workflow.

print.mocap_readiness *Print a MoCap readiness report*

Description

Print a MoCap readiness report

Usage

```
## S3 method for class 'mocap_readiness'
print(x, ...)
```

Arguments

x	A mocap_readiness object from assessMoCapReadiness().
...	Additional arguments (unused).

Value

Invisibly returns `x`.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`assessMoCapReadiness()` for computing the readiness report.

`print.multi_trial_phases` *Print method for multi-trial phases*

Description

Print method for multi-trial phases

Usage

```
## S3 method for class 'multi_trial_phases'
print(x, ...)
```

Arguments

`x`	A multi_trial_phases object
`...`	Additional arguments (unused)

`print.segmented_phases` *Print method for segmented phases*

Description

Print method for segmented phases

Usage

```
## S3 method for class 'segmented_phases'
print(x, ...)
```

Arguments

`x`	A segmented_phases object
`...`	Additional arguments (unused)

print.SkeletonModel *Print a SkeletonModel object*

Description

Print a SkeletonModel object

Usage

```
## S3 method for class 'SkeletonModel'
print(x, ...)
```

Arguments

x	A SkeletonModel object.
...	Additional arguments (ignored).

Value

Invisibly returns x.

print.waveform_pca *Print waveform PCA result*

Description

Print waveform PCA result

Usage

```
## S3 method for class 'waveform_pca'
print(x, ...)
```

Arguments

x	A waveform_pca object
...	Additional arguments (unused)

print.waveform_umap *Print waveform UMAP result*

Description

Print waveform UMAP result

Usage

```
## S3 method for class 'waveform_umap'
print(x, ...)
```

Arguments

x	A waveform_umap object
...	Additional arguments (unused)

processEMG *Process EMG for biomechanics workflows*

Description

Applies optional band-pass filtering, rectification, RMS envelope extraction, optional low-pass smoothing, and optional MVC normalization.

Usage

```
processEMG(
  x,
  sampling_rate,
  bandpass = c(20, 450),
  envelope_cutoff = 6,
  rms_window_ms = 50,
  mvc = NULL,
  filter_method = c("butterworth", "moving_average")
)
```

Arguments

x	Numeric vector or matrix (time x channels).
sampling_rate	Sampling rate in Hz.
bandpass	Optional length-2 numeric vector (Hz). If NULL, no band-pass filtering is applied.
envelope_cutoff	Low-pass cutoff (Hz) applied to RMS envelope.

rms_window_ms	RMS window length in milliseconds.
mvc	Optional MVC value(s) for normalization.
filter_method	Filter method used in low-pass steps.

Value

A list with filtered, rectified, envelope, and (when mvc is provided) normalized.

References

Merletti R, Parker PA (2004). "Electromyography: Physiology, Engineering, and Non-Invasive Applications." IEEE Press/Wiley.

See Also

rectifyEMG() for signal rectification, computeRMSEnvelope() for RMS envelope computation, normalizeEMG() for MVC or peak normalization, integrateEMGMoCap() for EMG-MoCap data integration.

Examples

```
set.seed(1)
emg <- matrix(rnorm(2000), ncol = 2)
out <- processEMG(emg, sampling_rate = 1000)
```

---

projectTo2D *Project 3D coordinates to a 2D plane*

Description

Selects two of three coordinate axes based on the anatomical plane, returning a data frame with u (horizontal) and v (vertical) columns.

Usage

```
projectTo2D(x, y, z, plane = c("sagittal", "frontal", "transverse"))
```

Arguments

x	Numeric vector of X (medial-lateral) coordinates.
y	Numeric vector of Y (anterior-posterior) coordinates.
z	Numeric vector of Z (vertical) coordinates.
plane	Character string specifying the projection plane. One of "sagittal", "frontal", or "transverse".

Value

A data frame with columns u and v.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`plotSkeleton()` for 2D skeleton visualization, `plotSkeleton3D()` for pseudo-3D skeleton rendering.

Examples

```
proj <- projectTo2D(c(1, 2), c(3, 4), c(5, 6), plane = "sagittal")
proj$u  # Y values (anterior-posterior)
proj$v  # Z values (vertical)
```

---

quaternionToEuler *Convert quaternion to Euler angles*

Description

Converts a quaternion (w, x, y, z) representation to Euler angles (roll, pitch, yaw) using the specified rotation order.

Usage

```
quaternionToEuler(w, x, y, z, order = "ZYX", degrees = TRUE)
```

Arguments

`w`	Numeric. Scalar (real) part of the quaternion.
`x`	Numeric. First imaginary component.
`y`	Numeric. Second imaginary component.
`z`	Numeric. Third imaginary component.
`order`	Character. Rotation order. Default `"ZYX"`.
`degrees`	Logical. If `TRUE` (default), return Euler angles in degrees. If `FALSE`, return in radians.

Details

For the `"ZYX"` (Tait-Bryan) convention:

- roll = rotation about X axis
- pitch = rotation about Y axis
- yaw = rotation about Z axis

The quaternion is assumed to be unit (normalized). If not unit, it is normalized internally before conversion.

Value

A matrix with columns `roll`, `pitch`, `yaw`. If inputs are scalars, returns a 1-row matrix. If inputs are vectors, returns a matrix with one row per element.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

Grood ES, Suntay WJ (1983). "A joint coordinate system for the clinical description of three-dimensional motions: application to the knee." Journal of Biomechanical Engineering, 105(2), 136-144.

See Also

`eulerToQuaternion()`, `calculateJointAngles()`, `vectorAngle()`

Examples

```
# Identity quaternion -> zero Euler angles
quaternionToEuler(1, 0, 0, 0)

# 90-degree rotation about Z axis
quaternionToEuler(cos(pi/4), 0, 0, sin(pi/4))
```

---

quickStartMoCap *Run a one-command beginner workflow*

Description

Runs a compact end-to-end pipeline for first-time users: kinematics derivatives, readiness scoring, and optional force-plate, inverse-dynamics, and EMG modules.

Usage

```
quickStartMoCap(
  n_frames = 300,
  sampling_rate = NULL,
  emg_sampling_rate = 1000,
  seed = 123,
  mocap = NULL,
  path = NULL,
  format = c("auto", "csv", "c3d", "trc", "bvh", "amc"),
  forces = NULL,
  joints = NULL,
  joint_angles = NULL,
  emg = NULL
)
```

Arguments

n_frames	Number of MoCap frames for demo mode.
sampling_rate	MoCap sampling rate in Hz. In non-demo mode, if NULL, uses samplingRate(mocap).
emg_sampling_rate	EMG sampling rate in Hz.
seed	Random seed for reproducibility in demo mode.
mocap	Optional PhysioExperiment object to analyze.
path	Optional file path to load via readMoCapAuto().
format	Format hint passed to readMoCapAuto() when path is used.
forces	Optional force matrix/data.frame for force-plate analysis.
joints	Optional joint-center data for inverse dynamics.
joint_angles	Optional joint-angle data for inverse dynamics.
emg	Optional EMG matrix for EMG processing.

Details

If mocap or path is omitted, synthetic demo data are generated.

Value

An object of class "mocap_quickstart" containing generated outputs, readiness report, and notes for skipped/failed optional modules.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

demoMoCapData() for generating demo data without analysis, assessMoCapReadiness() for data quality assessment, print.mocap_quickstart() for formatted display of results.

Examples

```
qs <- quickStartMoCap(seed = 1)
qs
```

readAMC *Read AMC Motion Capture File (.amc)*

Description

Parses an Acclaim Motion Capture (AMC) file containing frame-by-frame joint angle data. AMC files store joint angles per frame, with each frame listing joint names followed by their DOF values. An optional ASF skeleton can be provided for validation and DOF mapping.

Usage

```
readAMC(path, asf = NULL, fps = 120)
```

Arguments

path	Character string giving the path to the .amc file.
asf	An ASFSkeleton object (from readASF()), or NULL. If provided, used for DOF validation and enriched metadata.
fps	Numeric frame rate in Hz. AMC files do not store frame rate, so this must be specified (default: 120).

Value

A PhysioExperiment object with:

assays rotation_x, rotation_y, rotation_z for all joints; position_x, position_y, position_z for the root joint only.

colData DataFrame with label (joint names), type ("root" or "joint"), and dof_count (number of DOFs).

metadata List with asf_skeleton (if provided), source_file, and units (from ASF if provided).

samplingRate Set to fps.

References

CMU Graphics Lab (2003). "CMU Motion Capture Database." http://mocap.cs.cmu.edu/.

See Also

readASF() for reading skeleton definitions in ASF format, readMoCapCSV() for reading motion capture data from CSV files.

Examples

```
asf_file <- system.file("testdata", "sample.asf", package = "PhysioMoCap")
amc_file <- system.file("testdata", "sample.amc", package = "PhysioMoCap")
if (nzchar(asf_file) && nzchar(amc_file)) {
  skel <- readASF(asf_file)
  pe <- readAMC(amc_file, asf = skel)
  pe
}
```

---

readASF *Read ASF Skeleton Definition File (.asf)*

Description

Parses an Acclaim Skeleton File (ASF) that defines the skeleton hierarchy, bone properties, and degrees of freedom. ASF files contain sections for units, root configuration, bone data, and hierarchy relationships.

Usage

```
readASF(path)
```

Arguments

`path`	Character string giving the path to the `.asf` file.

Value

An S3 object of class `"ASFSkeleton"` with components:

units Named list of unit definitions (mass, length, angle).

root List with root joint configuration: position, orientation, order, and axis.

bones Named list of bone definitions, each with id, name, direction, length, axis, dof, and limits.

hierarchy Named list mapping parent joint names to character vectors of child joint names.

References

CMU Graphics Lab (2003). "CMU Motion Capture Database." http://mocap.cs.cmu.edu/.

See Also

`readAMC()` for reading motion data in AMC format, `print.ASFSkeleton()` for displaying skeleton structure.

Examples

```
asf_file <- system.file("testdata", "sample.asf", package = "PhysioMoCap")
if (nzchar(asf_file)) {
  skel <- readASF(asf_file)
  skel
}
```

`readBVH` *Read BVH Skeleton Animation File (.bvh)*

Description

Reads a BVH (Biovision Hierarchy) file containing skeleton animation data. BVH files have two sections: HIERARCHY (joint tree structure with offsets and channel definitions) and MOTION (frame data). The ROOT joint has 6 channels (3 position + 3 rotation), while child JOINTs have 3 channels (rotation only).

Usage

```
readBVH(path)
```

Arguments

`path`	Character string giving the path to the `.bvh` file.

Value

A `PhysioExperiment` object with rotation assays (`rotation_x`, `rotation_y`, `rotation_z`) for all joints, and position assays (`position_x`, `position_y`, `position_z`) for the root joint. `colData` includes joint names (`label`), channel type (`type`), and `parent_joint`. `metadata` includes `bvh_skeleton` (hierarchy tree), `rotation_order`, `frame_time`, `offsets`, and `source_file`. `samplingRate` is computed as `1 / frame_time`.

References

Meredith M, Maddock S (2001). "Motion Capture File Formats Explained." Department of Computer Science, University of Sheffield.

See Also

`readC3D()`, `readTRC()`, `readOpenPose()`

Examples

```
bvh_file <- system.file("testdata", "sample.bvh", package = "PhysioMoCap")
if (nzchar(bvh_file)) {
  pe <- readBVH(bvh_file)
  pe
}
```

readC3D *Read C3D Motion Capture File*

Description

Reads a C3D file containing 3D marker position data using the **c3dr** package. C3D is a widely used binary format for storing biomechanical motion capture data including point (marker) positions and optional analog channel data (e.g., force plate signals).

Usage

```
readC3D(path, include_analog = FALSE)
```

Arguments

path	Character string giving the path to the .c3d file.
include_analog	Logical; if TRUE, analog data (e.g., force plate channels) is extracted and stored in metadata(pe)$analog_data as a data frame. Default is FALSE.

Value

A PhysioExperiment object with three assays: "position_x", "position_y", and "position_z", each a matrix with rows as time frames and columns as markers. If the C3D file contains residual data, a "quality" assay is also included. Column metadata (colData) contains label (marker names from POINT:LABELS), type ("marker"), and body_segment (NA). Metadata includes c3d_parameters, source_file, and a time vector computed from frame rate.

References

C3D.org. "The C3D File Format." https://www.c3d.org/.

See Also

readTRC(), readBVH(), readOpenPose()

Examples

```
if (requireNamespace("c3dr", quietly = TRUE)) {
  c3d_file <- c3dr::c3d_example()
  pe <- readC3D(c3d_file)
  pe
}
```

`readDeepLabCut` *Read DeepLabCut output*

Description

Reads DeepLabCut (DLC) pose estimation output from CSV or HDF5 format and returns a PhysioExperiment object with keypoint coordinates and confidence scores.

Usage

```
readDeepLabCut(path, fps = 30, format = c("csv", "h5"))
```

Arguments

`path`	Path to a DeepLabCut output file (CSV or H5).
`fps`	Frame rate in Hz (frames per second). Default 30. DeepLabCut does not store frame rate in its output, so this must be specified by the user.
`format`	File format: `"csv"` or `"h5"`. Default `"csv"`.

Details

DeepLabCut outputs pose estimation results in CSV or HDF5 format.

CSV format has a 3-row multi-level header:

1. Row 1: scorer name (the DLC model name)
2. Row 2: bodypart names
3. Row 3: coordinate type (x, y, likelihood)

Followed by numeric data where each row is a frame and columns are grouped as `[x, y, likelihood]` triplets per bodypart.

H5 format stores data in a hierarchical HDF5 structure under `df_with_missing/table`. Requires the `rhdf5` package.

Value

A PhysioExperiment with assays:

keypoint_x X coordinates matrix (frames x bodyparts)

keypoint_y Y coordinates matrix (frames x bodyparts)

confidence Detection likelihood matrix (frames x bodyparts)

The `colData` contains columns `label` (bodypart names), `type` (`"keypoint"`), and `scorer` (the DLC scorer/model name).

The `metadata` list contains `dlc_scorer`, `format`, and `source_file`.

References

Mathis A, Mamidanna P, Cury KM, Abe T, Murthy VN, Mathis MW, Bethge M (2018). "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning." Nature Neuroscience, 21(9), 1281-1289.

See Also

`readOpenPose()`, `readMediaPipe()`, `readOpenCap()`

Examples

```
## Not run:
# Read DeepLabCut CSV output
pe <- readDeepLabCut("path/to/DLC_output.csv", fps = 30)

# Read HDF5 format
pe <- readDeepLabCut("path/to/DLC_output.h5", fps = 25, format = "h5")

## End(Not run)
```

---

readMediaPipe *Read MediaPipe landmark output*

Description

Reads MediaPipe landmark data from a directory of per-frame JSON files or a single CSV file and returns a PhysioExperiment object.

Usage

```
readMediaPipe(path, model = c("pose", "hand"), fps = 30)
```

Arguments

`path`	Path to a directory of JSON files or a single CSV file.
`model`	Landmark model: `"pose"` (33 landmarks) or `"hand"` (21 landmarks). Default `"pose"`.
`fps`	Frame rate in Hz (frames per second). Default 30.

Details

MediaPipe can output landmarks in two formats:

JSON (directory of per-frame files): Each JSON file contains a `"landmarks"` array where each element has `x`, `y`, `z`, and `visibility` fields. Optionally, a `"world_landmarks"` array may be present with world-space coordinates.

CSV (single file): Columns named `landmark_0_x`, `landmark_0_y`, `landmark_0_z`, `landmark_0_visibility`, `landmark_1_x`, etc. Each row corresponds to one frame.

Pose model (33 landmarks): Full body landmarks from nose to feet.

Hand model (21 landmarks): Hand landmarks from wrist to fingertips.

Value

A PhysioExperiment with assays:

landmark_x X coordinates matrix (frames x landmarks)

landmark_y Y coordinates matrix (frames x landmarks)

landmark_z Z coordinates matrix (frames x landmarks)

visibility Visibility scores matrix (frames x landmarks)

If world landmarks are present (JSON with `world_landmarks` field), additional assays are included:

world_x World X coordinates matrix

world_y World Y coordinates matrix

world_z World Z coordinates matrix

The `colData` contains columns `label` (landmark name), `type` (`"landmark"`), `model` (the MediaPipe model used), and `landmark_idx` (0-based landmark index).

References

Lugaresi C, Tang J, Nash H, McClanahan C, Uboweja E, Hays M, Zhang F, Chang CL, Yong MG, Lee J, et al. (2019). "MediaPipe: A Framework for Building Perception Pipelines." arXiv:1906.08172.

See Also

`readOpenPose()`, `readDeepLabCut()`, `define_skeleton()`

Examples

```
## Not run:
# Read directory of MediaPipe JSON files
pe <- readMediaPipe("path/to/mediapipe_output/", model = "pose", fps = 30)

# Read CSV format
pe <- readMediaPipe("path/to/landmarks.csv", model = "hand", fps = 60)

## End(Not run)
```

`readMoCapAuto` *Read motion-capture files with automatic format detection*

Description

Chooses a reader based on file extension and returns a `PhysioExperiment`. This is designed for first-time users who want one entry point for common MoCap formats.

Usage

```
readMoCapAuto(
  path,
  format = c("auto", "csv", "c3d", "trc", "bvh", "amc"),
  sampling_rate = NULL,
  sep = ",",
  header_rows = 1L,
  skip = 0L,
  include_analog = FALSE,
  asf = NULL,
  fps = 120
)
```

Arguments

`path`	Path to a motion-capture file.
`format`	Reader format. `"auto"` detects from extension (`.c3d`, `.trc`, `.csv`, `.tsv`, `.bvh`, `.amc`).
`sampling_rate`	Sampling rate for CSV files when not inferable.
`sep`	Delimiter used for CSV/TSV files.
`header_rows`	Number of header rows for CSV/TSV.
`skip`	Number of lines to skip before data for CSV/TSV.
`include_analog`	Logical; passed to `readC3D()`.
`asf`	Optional ASF skeleton for AMC files. Either an `ASFSkeleton` object or a path to an `.asf` file.
`fps`	Frame rate for AMC files.

Value

A `PhysioExperiment` object.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

readMoCapCSV() for CSV/TSV motion capture data, readASF() and readAMC() for Acclaim skeleton and motion files, assessMoCapReadiness() for data quality assessment.

Examples

```
trc_file <- system.file("testdata", "sample.trc", package = "PhysioMoCap")
if (nzchar(trc_file)) {
  pe <- readMoCapAuto(trc_file)
  pe
}
```

---

readMoCapCSV *Read Motion Capture CSV Data*

Description

Generic CSV reader that handles common motion capture CSV exports from various systems including Qualisys, Vicon, and generic marker position formats. Returns a PhysioExperiment with position_x, position_y, and position_z assays.

Usage

```
readMoCapCSV(
  path,
  format = c("auto", "xyz", "wide", "long", "qualisys", "vicon"),
  sampling_rate = NULL,
  header_rows = 1L,
  skip = 0L,
  sep = ",",
  marker_names = NULL,
  coord_columns = NULL
)
```

Arguments

path	Path to the CSV file.
format	Format hint: "auto" (detect), "xyz" (columns like marker1_x, marker1_y, marker1_z), "wide" (columns like Time, M1X, M1Y, M1Z), "long" (columns: frame, marker, x, y, z), "qualisys" (Qualisys TSV export), or "vicon" (Vicon CSV export). Default "auto".
sampling_rate	Sampling rate in Hz. Required if not detectable from the file (e.g., from a Time column). If NULL and not detectable, an error is raised.
header_rows	Number of header rows. Default 1.
skip	Number of lines to skip before reading. Default 0.
sep	Column separator. Default ",".

`marker_names` Explicit marker names (overrides auto-detection from column names). Must match the number of markers detected from columns.

`coord_columns` Mapping of coordinate types as a named list of regex patterns, e.g., `list(x = "_X$", y = "_Y$", z = "_Z$")`. Used only when `format` is `"wide"` or `"auto"`.

Details

Auto-detection logic:

When `format = "auto"`, the function inspects column names:

1. If columns match `*_x`, `*_y`, `*_z` pattern (case-insensitive) with consistent marker prefixes, the `"xyz"` format is used.
2. If columns match `*X`, `*Y`, `*Z` suffix pattern, the `"wide"` format is used.
3. If the header contains "Qualisys" or "QTM", the `"qualisys"` format is used.
4. If columns include `frame`, `marker`, `x`, `y`, `z` (case-insensitive), the `"long"` format is used.

If a `Time` or `time` column is present and contains numeric values, the sampling rate is computed from the median time step. An explicit `sampling_rate` argument always takes precedence.

Value

A PhysioExperiment with assays:

position_x X coordinates matrix (frames x markers)

position_y Y coordinates matrix (frames x markers)

position_z Z coordinates matrix (frames x markers)

The `colData` contains columns `label` (marker names) and `type` (`"marker"`). The `metadata` list contains `format`, `source_file`, and optionally `time` (if a Time column was found).

References

Wickham H (2014). "Tidy Data." Journal of Statistical Software, 59(10), 1-23.

See Also

`readASF()` and `readAMC()` for Acclaim skeleton and motion files, `readMoCapAuto()` for automatic format detection.

Examples

```
## Not run:
# Read xyz-format CSV
pe <- readMoCapCSV("markers.csv", sampling_rate = 120)

# Read with auto-detection from Time column
pe <- readMoCapCSV("markers.csv")

# Read tab-separated file
pe <- readMoCapCSV("qualisys_export.tsv", format = "qualisys", sep = "\t")
```

```
# Override marker names
pe <- readMoCapCSV("data.csv", marker_names = c("Hip", "Knee", "Ankle"),
                   sampling_rate = 100)

## End(Not run)
```

---

readMOT *Read OpenSim Motion File (.mot)*

Description

Reads an OpenSim motion file containing joint angles, kinematics, or other time-series data. MOT files use a tab-separated format with a header block ending in "endheader".

Usage

```
readMOT(path)
```

Arguments

path	Character string giving the path to the .mot file.

Value

A PhysioExperiment object with the data stored in the "raw" assay. The first column (time) is used to compute the sampling rate. Header metadata such as inDegrees is stored in metadata().

References

Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, Guendelman E, Thelen DG (2007). "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement." IEEE Transactions on Biomedical Engineering, 54(11), 1940-1950.

See Also

readSTO(), readTRC(), readOpenCap()

Examples

```
mot_file <- system.file("testdata", "sample.mot", package = "PhysioMoCap")
if (nzchar(mot_file)) {
  pe <- readMOT(mot_file)
  pe
}
```

readOpenCap *Read Data from OpenCap Cloud Platform*

Description

Downloads motion capture session data from the OpenCap API (https://app.opencap.ai). Marker trajectory data is returned as TRC format and kinematics data as MOT format, both parsed using the existing readTRC and readMOT functions.

Usage

```
readOpenCap(
  session_id,
  trial_id = NULL,
  api_key = NULL,
  data_type = c("markers", "kinematics"),
  base_url = "https://app.opencap.ai/api"
)
```

Arguments

session_id	Character string giving the OpenCap session identifier (the 36-character UUID at the end of the session URL).
trial_id	Character string giving the trial identifier within the session. If NULL (default), the first trial in the session is used.
api_key	Character string with the OpenCap API key. If NULL (default), the key is read from the OPENCAP_API_KEY environment variable.
data_type	Character string specifying the type of data to download. One of "markers" (TRC marker trajectories) or "kinematics" (MOT inverse kinematics results). Default is "markers".
base_url	Character string giving the base URL for the OpenCap API. Default is "https://app.opencap.ai/api".

Details

The function requires the **httr** package for HTTP requests. If **httr** is not installed, a clear error message is given.

Authentication uses an API key passed via the api_key parameter or the OPENCAP_API_KEY environment variable. The key is sent as a Bearer token in the Authorization header.

The download workflow is:

1. Retrieve session metadata from GET /sessions/{session_id}/
2. List trials from GET /sessions/{session_id}/trials/
3. Download the result file (TRC or MOT) for the selected trial
4. Parse using readTRC() or readMOT()

Value

A PhysioExperiment object. For data_type = "markers", this contains position_x, position_y, and position_z assays (from readTRC). For data_type = "kinematics", this contains a raw assay with joint angle data (from readMOT). Session and trial metadata are stored in metadata().

References

Uhlrich SD, Falisse A, Kidzinski L, Muccini J, Ko M, Chaudhari AS, Hicks JL, Delp SL (2023). "OpenCap: Human movement dynamics from smartphone videos." PLoS Computational Biology, 19(10), e1011462.

See Also

readTRC(), readMOT(), readC3D()

Examples

```
## Not run:
# Download marker data (requires API key)
pe <- readOpenCap("abcd1234-5678-90ab-cdef-1234567890ab")

# Download kinematics with explicit API key
pe <- readOpenCap(
  session_id = "abcd1234-5678-90ab-cdef-1234567890ab",
  api_key = "my-api-key",
  data_type = "kinematics"
)

## End(Not run)
```

---

readOpenPose *Read OpenPose JSON output*

Description

Reads OpenPose JSON keypoint data from a directory of frame files or a single JSON file and returns a PhysioExperiment object.

Usage

```
readOpenPose(path, model = c("BODY_25", "COCO"), fps = 30, person_id = 1L)
```

Arguments

path	Path to a directory of JSON files or a single JSON file.
model	Keypoint model: "BODY_25" (25 keypoints) or "COCO" (18 keypoints). Default "BODY_25".
fps	Frame rate in Hz (frames per second). Default 30.
person_id	Which person to extract (1-based index). Default 1 (first detected person).

Details

OpenPose outputs one JSON file per video frame. Each file contains a `"people"` array where each person's pose is stored as a flat array of `[x, y, confidence]` triplets.

BODY_25 model (25 keypoints): Nose, Neck, RShoulder, RElbow, RWrist, LShoulder, LElbow, LWrist, MidHip, RHip, RKnee, RAnkle, LHip, LKnee, LAnkle, REye, LEye, REar, LEar, LBigToe, LSmallToe, LHeel, RBigToe, RSmallToe, RHeel.

COCO model (18 keypoints): Nose, Neck, RShoulder, RElbow, RWrist, LShoulder, LElbow, LWrist, RHip, RKnee, RAnkle, LHip, LKnee, LAnkle, REye, LEye, REar, LEar.

Frames where the specified person is not detected will contain `NA` values for all keypoints.

Value

A PhysioExperiment with assays:

keypoint_x X coordinates matrix (frames x keypoints)

keypoint_y Y coordinates matrix (frames x keypoints)

confidence Detection confidence matrix (frames x keypoints)

The `colData` contains columns `label` (keypoint name), `type` (`"keypoint"`), and `model` (the OpenPose model used).

References

Cao Z, Hidalgo G, Simon T, Wei SE, Sheikh Y (2019). "OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields." IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(1), 172-186.

See Also

`readDeepLabCut()`, `readMediaPipe()`, `define_skeleton()`

Examples

```
## Not run:
# Read directory of OpenPose JSON files
pe <- readOpenPose("path/to/openpose_output/", fps = 30)

# Read with COCO model
pe <- readOpenPose("path/to/output/", model = "COCO", fps = 25)

# Extract second person
pe <- readOpenPose("path/to/output/", person_id = 2)

## End(Not run)
```

readOpenSimOutputs *Read OpenSim Output Files*

Description

Reads one or more OpenSim output files and returns PhysioExperiment objects.

Usage

```
readOpenSimOutputs(files, format = c("auto", "mot", "sto", "trc"))
```

Arguments

files	Character vector of file paths.
format	One of "auto", "mot", "sto", "trc".

Value

Named list of PhysioExperiment objects.

See Also

readMOT(), readSTO(), readTRC()

readSTO *Read OpenSim Storage File (.sto)*

Description

Reads an OpenSim storage file containing forces, moments, or other computed quantities. STO files share the same tab-separated format as MOT files with a header block ending in "endheader".

Usage

```
readSTO(path)
```

Arguments

path	Character string giving the path to the .sto file.

Value

A PhysioExperiment object with the data stored in the "raw" assay. The first column (time) is used to compute the sampling rate. Header metadata is stored in metadata().

References

Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, Guendelman E, Thelen DG (2007). "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement." IEEE Transactions on Biomedical Engineering, 54(11), 1940-1950.

See Also

`readMOT()`, `readTRC()`, `readOpenCap()`

Examples

```
sto_file <- system.file("testdata", "sample.sto", package = "PhysioMoCap")
if (nzchar(sto_file)) {
  pe <- readSTO(sto_file)
  pe
}
```

---

readTRC *Read OpenSim TRC File (.trc)*

Description

Reads marker trajectory data from an OpenSim TRC file. TRC files store 3D marker positions (X, Y, Z) in a specific header format that differs from MOT/STO files.

Usage

```
readTRC(path)
```

Arguments

`path`	Character string giving the path to the `.trc` file.

Value

A `PhysioExperiment` object with three assays: `"position_x"`, `"position_y"`, and `"position_z"`, each with columns named by marker. Header metadata (DataRate, Units, etc.) is stored in `metadata()`.

References

Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, Guendelman E, Thelen DG (2007). "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement." IEEE Transactions on Biomedical Engineering, 54(11), 1940-1950.

See Also

`readMOT()`, `readSTO()`, `readC3D()`

Examples

```
trc_file <- system.file("testdata", "sample.trc", package = "PhysioMoCap")
if (nzchar(trc_file)) {
  pe <- readTRC(trc_file)
  pe
}
```

`readVenus3D` *Read Venus3D CSV Data*

Description

Reads motion capture data exported from the OptiTrack -> Motive -> Venus3D pipeline. The Venus3D CSV format uses #-prefixed header lines for metadata and stores 3D marker coordinates in wide format with columns like `1(X)`, `1(Y)`, `1(Z)`, `2(X)`, etc.

Usage

```
readVenus3D(path, marker_names = NULL)
```

Arguments

`path`	Path to the Venus3D CSV file.
`marker_names`	Optional character vector of marker names to assign. Must match the number of markers in the file. If `NULL` (default), markers are labelled `P1`, `P2`, ... based on the Point Type header, or `M1`, `M2`, ... if that header is absent.

Details

Because Venus3D randomly reassigns marker labels across frames, downstream use of `trackMarkers()` is typically required to establish consistent marker identities.

Value

A PhysioExperiment with assays:

position_x X coordinates matrix (frames x markers)

position_y Y coordinates matrix (frames x markers)

position_z Z coordinates matrix (frames x markers)

The `colData` contains columns `label` (marker names) and `type` (`"marker"`). The `metadata` list contains `format`, `source_file`, `format_version`, `units`, `coordinate_system`, and `time` (numeric vector of frame times).

References

Venus3D Software Documentation, C-Motion Inc.

See Also

trackMarkers() for resolving Venus3D's random label assignment, readMoCapCSV() for generic CSV formats, readMoCapAuto() for automatic format detection.

Examples

```
## Not run:
pe <- readVenus3D("capture.csv")

# Assign meaningful marker names
pe <- readVenus3D("capture.csv",
                  marker_names = c("Hip", "Knee", "Ankle"))

# Follow with marker tracking to resolve label shuffling
pe_tracked <- trackMarkers(pe)

## End(Not run)
```

---

reconstructFPCA *Reconstruct waveforms from fPCA*

Description

Reconstructs individual waveforms using a subset of principal components.

Usage

```
reconstructFPCA(fpca_result, n_components = NULL, observation = NULL)
```

Arguments

fpca_result	An fpca_result object from fPCA().
n_components	Number of components to use for reconstruction.
observation	Indices of observations to reconstruct. If NULL, all.

Value

Matrix of reconstructed waveforms (time x observations).

References

Ramsay JO, Silverman BW (2005). "Functional Data Analysis." 2nd ed. Springer.

See Also

fPCA() for performing the decomposition, plotFPCA() for visualizing fPCA results.

Examples

```
# Create sample data and run fPCA first
set.seed(123)
t <- seq(0, 100, length.out = 100)
base_curve <- sin(2 * pi * t / 100) * 30
data <- sapply(1:20, function(i) base_curve * rnorm(1, 1, 0.2) + rnorm(100, 0, 2))
fpca_result <- fPCA(data, n_components = 4)

# Reconstruct using only first 2 PCs
reconstructed <- reconstructFPCA(fpca_result, n_components = 2)
```

---

rectifyEMG *Rectify EMG signals*

Description

Rectify EMG signals

Usage

```
rectifyEMG(x, method = c("fullwave", "halfwave"))
```

Arguments

x	Numeric vector or matrix (time x channels).
method	Rectification method: "fullwave" or "halfwave".

Value

Rectified signal with the same dimensions as x.

References

Merletti R, Parker PA (2004). "Electromyography: Physiology, Engineering, and Non-Invasive Applications." IEEE Press/Wiley.

See Also

computeRMSEnvelope() for computing RMS envelope, processEMG() for complete EMG processing pipeline.

Examples

```
x <- c(-1, -0.5, 0, 0.5, 1)
rectifyEMG(x, method = "fullwave")
```

`registerCurves` *Curve registration (time warping)*

Description

Aligns waveforms by estimating and removing phase variation. Uses landmark registration or continuous registration.

Usage

```
registerCurves(
  x,
  method = c("continuous", "landmark"),
  landmarks = NULL,
  template = NULL
)
```

Arguments

`x`	A PhysioExperiment object or matrix (time x observations).
`method`	Registration method: "landmark" or "continuous".
`landmarks`	For landmark method, matrix of landmark times (landmarks x obs).
`template`	Template curve to align to. If NULL, uses mean.

Details

Phase variation (timing differences) can obscure amplitude differences in biomechanical data. Registration separates phase and amplitude variation.

Value

A list containing:

`registered`	Registered waveforms
`warping`	Warping functions
`template`	Template used for registration

References

Ramsay JO, Silverman BW (2005). "Functional Data Analysis." 2nd ed. Springer.

See Also

`fPCA()` for functional PCA after registration, `plotGaitCycle()` for plotting registered gait waveforms.

Examples

```
# Simulate data with phase variation
set.seed(123)
t <- seq(0, 100, length.out = 100)

data <- sapply(1:20, function(i) {
  phase_shift <- rnorm(1, 0, 10)
  t_shifted <- t + phase_shift
  sin(2 * pi * t_shifted / 100) * 30 + rnorm(100, 0, 2)
})

pe <- PhysioExperiment(assays = list(values = data), samplingRate = 100)
reg_result <- registerCurves(pe, method = "continuous")
```

---

removeGravity *Remove gravity from accelerometer data*

Description

Subtracts the rotated gravity vector from raw accelerometer data, leaving only dynamic (linear) acceleration. The gravity direction is determined by rotating the reference gravity vector `(0, 0, -g)` from the world frame into the sensor frame using the provided orientation quaternions.

Usage

```
removeGravity(accel, orientation, g = 9.81)
```

Arguments

`accel`	Numeric matrix (n x 3) of raw accelerometer readings in m/s^2.
`orientation`	A data.frame or matrix containing orientation quaternions. If a data.frame, must contain columns `q_w`, `q_x`, `q_y`, `q_z` (as returned by `estimateOrientation()`). If a matrix, must have 4 columns (w, x, y, z).
`g`	Numeric. Gravitational acceleration magnitude (default 9.81 m/s^2).

Details

For each time step, the gravity vector in the world frame `(0, 0, -g)` is rotated into the sensor frame using the conjugate of the orientation quaternion. This rotated gravity is then subtracted from the raw accelerometer reading.

Value

Numeric matrix (n x 3) of dynamic (gravity-free) acceleration.

References

Madgwick SOH, Harrison AJL, Vaidyanathan R (2011). "Estimation of IMU and MARG orientation using a gradient descent algorithm." IEEE International Conference on Rehabilitation Robotics.

See Also

estimateOrientation(), calibrateIMU(), quaternionToEuler()

Examples

```
n <- 100
accel <- matrix(c(rep(0, n), rep(0, n), rep(-9.81, n)), ncol = 3)
gyro <- matrix(0, nrow = n, ncol = 3)
ori <- estimateOrientation(accel, gyro, sampling_rate = 100)
dyn_accel <- removeGravity(accel, ori)
```

---

reportGaps *Report summary statistics for detected gaps*

Description

Prints a human-readable summary of gap statistics including total number of gaps, average gap size, and percentage of missing data.

Usage

```
reportGaps(gaps, sampling_rate)
```

Arguments

gaps	A data.frame as returned by detectGaps
sampling_rate	Sampling rate in Hz (used to report gap durations)

Value

Invisibly returns a list with summary statistics:

- total_gaps - Total number of gaps
- avg_size - Average gap size in samples
- total_missing - Total number of missing samples
- pct_missing - Percentage of missing data (if computable)

References

Federolf PA (2013). "A novel approach to solve the 'missing marker problem' in marker-based motion analysis that does not require additional assumptions about the biodynamic model." Journal of Biomechanics, 46(13), 2173-2178.

See Also

detectGaps(), fillGaps(), fillGapsLinear()

Examples

```
gaps <- data.frame(
  channel = c("M1", "M1", "M2"),
  start = c(10, 50, 20),
  end = c(15, 55, 30),
  size = c(6, 6, 11)
)
reportGaps(gaps, sampling_rate = 120)
```

---

resampleSignal *Resample signal assays to a new sampling rate*

Description

Resamples all (or selected) numeric assays of a PhysioExperiment object to a new target sampling rate. Both upsampling and downsampling are supported.

Usage

```
resampleSignal(
  pe,
  target_rate,
  method = c("linear", "spline"),
  assay_names = NULL
)
```

Arguments

pe	A PhysioExperiment object.
target_rate	Target sampling rate in Hz.
method	Interpolation method: "linear" (default) or "spline".
assay_names	Character vector of assay names to resample. If NULL (default), all numeric assays are resampled.

Value

A new PhysioExperiment with resampled assays, updated `samplingRate`, and updated `metadata$time` vector.

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

`resampleVector()` for resampling individual numeric vectors, `synchronizeSignals()` for synchronizing multiple experiments.

Examples

```
pe <- PhysioCore::PhysioExperiment(
  assays = S4Vectors::SimpleList(
    position_x = matrix(sin(seq(0, 2 * pi, length.out = 100)), ncol = 1)
  ),
  colData = S4Vectors::DataFrame(label = "M1", type = "marker"),
  samplingRate = 100
)
pe2 <- resampleSignal(pe, target_rate = 200)
PhysioCore::samplingRate(pe2)  # 200
```

---

resampleVector *Resample a single numeric vector*

Description

Resamples a numeric vector from one sampling rate to another using interpolation.

Usage

```
resampleVector(x, from_rate, to_rate, method = c("linear", "spline"))
```

Arguments

x	Numeric vector to resample.
from_rate	Original sampling rate in Hz.
to_rate	Target sampling rate in Hz.
method	Interpolation method: "linear" uses approx, "spline" uses spline.

Value

A numeric vector resampled at the target rate.

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

resampleSignal() for resampling PhysioExperiment objects, synchronizeSignals() for synchronizing multiple signals.

Examples

```
# Resample a 100Hz signal to 200Hz
x <- sin(seq(0, 2 * pi, length.out = 100))
x_up <- resampleVector(x, from_rate = 100, to_rate = 200)
length(x_up)  # 200
```

run_opensim_toolchain *Run OpenSim CLI Toolchain via PhysioOpenSim*

Description

Executes OpenSim setup XML files sequentially through PhysioOpenSim.

Usage

```
run_opensim_toolchain(
  ik_setup = NULL,
  id_setup = NULL,
  so_setup = NULL,
  rra_setup = NULL,
  cmc_setup = NULL,
  analyze_setup = NULL,
  workdir = NULL,
  cli = NULL,
  timeout_sec = 0L,
  fail_on_error = TRUE,
  extra_args = character()
)
```

Arguments

ik_setup	Optional path to IK setup XML.
id_setup	Optional path to ID setup XML.
so_setup	Optional path to SO setup XML.
rra_setup	Optional path to RRA setup XML.
cmc_setup	Optional path to CMC setup XML.
analyze_setup	Optional path to Analyze setup XML.
workdir	Optional working directory for all tool runs.
cli	Optional OpenSim CLI command/path.
timeout_sec	Timeout in seconds for each tool execution.
fail_on_error	If TRUE, abort on non-zero exit status.
extra_args	Optional character vector appended to each tool run.

Value

Named list with entries for executed tools (ik, id, so, rra, cmc, analyze).

See Also

batch_analyze_opensim(), create_schema_from_opensim()

`runBenchmarkSuite` *Run a benchmark suite from a manifest*

Description

Executes all benchmark rows in a manifest and returns aggregate summaries.

Usage

```
runBenchmarkSuite(
  manifest,
  data_dir = ".",
  thresholds = defaultBenchmarkThresholds("balanced"),
  alignment = c("truncate", "resample"),
  report_dir = NULL
)
```

Arguments

`manifest`	Manifest data.frame or manifest CSV path. `prediction_file`/`reference_file` can point to `.csv`, `.mot`, `.sto`, or `.trc` files.
`data_dir`	Base directory for relative manifest file paths.
`thresholds`	Named threshold list from `defaultBenchmarkThresholds()`.
`alignment`	Alignment mode passed to `benchmarkAgreement()`.
`report_dir`	Optional output directory. If supplied, summary and detailed CSV reports are written.

Value

Object of class `"benchmark_suite"` with `summary`, `metrics`, `manifest`, and `thresholds`.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

See Also

`benchmarkAgreement()` for single-trial agreement analysis, `createBenchmarkExample()` for generating example benchmark data, `print.benchmark_suite()` for displaying suite results.

Examples

```
ex <- createBenchmarkExample(n_trials = 2, seed = 1)
suite <- runBenchmarkSuite(ex$manifest, data_dir = ex$data_dir)
suite$summary
```

savgolFilter *Savitzky-Golay filter for smoothing and differentiation*

Description

Pure R implementation of the Savitzky-Golay filter using local polynomial fitting. Useful for smoothing noisy signals while preserving features like peak height and width better than simple moving averages.

Usage

```
savgolFilter(x, window_length = 11, poly_order = 3, deriv = 0)
```

Arguments

x	A numeric vector or matrix (time x channels).
window_length	Window length for the filter. Must be a positive odd integer.
poly_order	Polynomial order for local fitting (must be less than window_length).
deriv	Derivative order. 0 for smoothing, 1 for first derivative, 2 for second derivative, etc.

Details

The Savitzky-Golay filter fits a local polynomial of degree poly_order to a window of window_length points, using least-squares. The filter coefficients are computed analytically and applied via convolution.

For differentiation (deriv > 0), the result is the derivative of the fitted polynomial, which provides a smooth estimate of the derivative.

Edge handling: the first and last floor(window_length/2) points are computed with truncated windows where possible, or set to NA.

Value

Filtered/smoothed data with the same dimensions as x.

References

Savitzky A, Golay MJE (1964). "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." Analytical Chemistry, 36(8), 1627-1639.

See Also

butterworthFilter() for frequency-domain Butterworth filtering, movingAverage() for simple moving average smoothing, differentiate() for numerical differentiation.

Examples

```
# Smooth a noisy sine wave
t <- seq(0, 2 * pi, length.out = 200)
x <- sin(t) + rnorm(200, sd = 0.2)
x_smooth <- savgolFilter(x, window_length = 11, poly_order = 3)

# Compute first derivative
dx <- savgolFilter(x, window_length = 11, poly_order = 3, deriv = 1)
```

---

schema_balance *Pre-built Balance Schema*

Description

Task schema for postural control/balance analysis.

Usage

```
schema_balance
```

Format

A TaskSchema object for continuous balance assessment.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), schema_gait, schema_cutting

Examples

```
print(schema_balance)
```

schema_cutting *Pre-built Cutting/Change of Direction Schema*

Description

Task schema for cutting and change of direction analysis.

Usage

```
schema_cutting
```

Format

A TaskSchema object with COD-specific events and phases.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), schema_gait, schema_running

Examples

```
print(schema_cutting)
```

schema_cycling *Pre-built Cycling Schema*

Description

Task schema for cycling pedal stroke analysis.

Usage

```
schema_cycling
```

Format

A TaskSchema object with cycling-specific events and phases.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), schema_gait, schema_running

Examples

```
print(schema_cycling)
```

---

schema_gait *Pre-built Gait Cycle Schema*

Description

Task schema for walking gait analysis with standard events and phases.

Usage

```
schema_gait
```

Format

A TaskSchema object with:

events Heel strike (x2), foot flat, midstance, heel off, toe off

phases Stance (with loading, midstance, propulsion subphases), Swing

normalization cycle (0-100%)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), getSchema(), schema_running

Examples

```
print(schema_gait)
getEventNames(schema_gait)
getPhaseNames(schema_gait)
```

schema_jump *Pre-built Jump Schema*

Description

Task schema for vertical jump analysis (countermovement, drop jump, etc.).

Usage

```
schema_jump
```

Format

A TaskSchema object with jump-specific events and phases.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), schema_gait, schema_throw

Examples

```
print(schema_jump)
```

schema_running *Pre-built Running Schema*

Description

Task schema for running gait analysis.

Usage

```
schema_running
```

Format

A TaskSchema object with running-specific events and phases.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), schema_gait, schema_jump

Examples

```
print(schema_running)
```

---

schema_throw *Pre-built Throwing Schema*

Description

Task schema for throwing motion analysis (baseball, softball, etc.).

Usage

```
schema_throw
```

Format

A TaskSchema object with throwing-specific events and phases.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), schema_gait, schema_cutting

Examples

```
print(schema_throw)
```

`segmentParameters` *Get body segment inertial parameters (BSIP)*

Description

Returns a data.frame of body segment inertial parameters including mass fractions and center of mass proximal fractions for standard anthropometric models.

Usage

```
segmentParameters(model = "deLeva_male")
```

Arguments

`model`	Character string specifying the anthropometric model. One of `"deLeva_male"`, `"deLeva_female"`, or `"winter"`. Default is `"deLeva_male"`.

Details

The De Leva (1996) tables provide adjusted body segment parameters based on Zatsiorsky's data, separately for males and females. The Winter (2009) model provides a simplified set of parameters commonly used in gait analysis.

Segments include: head, trunk, upper_arm_r, upper_arm_l, forearm_r, forearm_l, hand_r, hand_l, thigh_r, thigh_l, shank_r, shank_l, foot_r, foot_l (14 segments total).

Value

A `data.frame` with columns:

segment Character, name of the body segment.

mass_fraction Numeric, fraction of total body mass (as percentage, 0-100).

com_proximal_fraction Numeric, fraction of segment length from the proximal endpoint to the segment center of mass (0-1).

proximal_marker Character, default proximal marker name.

distal_marker Character, default distal marker name.

References

De Leva, P. (1996). Adjustments to Zatsiorsky-Seluyanov's segment inertia parameters. *Journal of Biomechanics*, 29(9), 1223-1230.

Winter, D.A. (2009). *Biomechanics and Motor Control of Human Movement* (4th ed.). Wiley.

See Also

`calculateCOM()` for whole-body center of mass computation, `calculateSegmentCOM()` for individual segment center of mass, `estimateSegmentInertia()` for segment inertial properties.

Examples

```
bsip <- segmentParameters("deLeva_male")
head(bsip)
sum(bsip$mass_fraction)  # approximately 100
```

---

segmentPhases *Segment data into phases*

Description

Divides movement data into phases based on detected events and schema definition.

Usage

```
segmentPhases(x, events, schema, include_subphases = TRUE)
```

Arguments

x	PhysioExperiment object or matrix (time x channels)
events	A detected_events data.frame from detectEvents()
schema	TaskSchema object defining phase structure
include_subphases	Whether to include subphases in output

Value

A segmented_phases object (list) containing:

- phases - List of phase data
- metadata - Phase timing information
- schema - Original schema

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

detectEvents(), extractPhase(), phaseTiming(), normalizeMovement()

Examples

```
# Assuming events have been detected
# phases <- segmentPhases(data, events, schema_gait)
```

sem *Standard Error of Measurement (SEM)*

Description

Computes the standard error of measurement from the SD of scores and a reliability coefficient (ICC or test-retest correlation).

Usage

```
sem(x, icc_value = NULL, reliability = NULL)
```

Arguments

x	Numeric vector of observed scores. Used to compute SD.
icc_value	Numeric; ICC value to use as the reliability coefficient.
reliability	Numeric; alternative reliability coefficient (e.g., test-retest correlation). Exactly one of icc_value or reliability must be provided.

Details

SEM is computed as:

$$SEM = SD \times \sqrt{1 - r}$$

where r is the reliability coefficient (ICC or correlation).

Value

Numeric SEM value.

References

Shrout PE, Fleiss JL (1979). "Intraclass Correlations: Uses in Assessing Rater Reliability." Psychological Bulletin, 86(2), 420-428.

See Also

icc() for computing intraclass correlation coefficients, mdc() for minimal detectable change based on SEM.

Examples

```
scores <- c(10, 12, 15, 11, 13, 14, 9, 16, 12, 11)
sem(scores, icc_value = 0.90)
```

`SkeletonModel` *Create a SkeletonModel object*

Description

Constructs an S3 `SkeletonModel` object that defines a skeleton topology for pose estimation or motion capture data. A skeleton consists of named keypoints (joints/markers), bones connecting them, and an optional hierarchical tree structure.

Usage

```
SkeletonModel(name, keypoints, bones, hierarchy, root_keypoint)
```

Arguments

`name`	Character string identifying the skeleton model (e.g., `"BODY_25"`, `"COCO"`).
`keypoints`	A `data.frame` with columns: **id** Integer, 0-indexed keypoint identifier. **label** Character, human-readable keypoint name. **body_region** Character, anatomical region (e.g., `"head"`, `"torso"`, `"left_arm"`).
`bones`	A `data.frame` with columns: **from_id** Integer, source keypoint id. **to_id** Integer, target keypoint id. **bone_name** Character, descriptive name for the bone segment.
`hierarchy`	Named list of parent-to-children relationships. Each element name is a parent keypoint label; the value is a character vector of child keypoint labels.
`root_keypoint`	Character, label of the root keypoint in the hierarchy (e.g., `"MidHip"` for BODY_25).

Value

A `SkeletonModel` object (S3 class).

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`define_skeleton()`, `get_bone_connections()`, `get_segment_lengths()`

Examples

```
# Minimal skeleton
kp <- data.frame(
  id = 0:2,
  label = c("Head", "Torso", "Hip"),
  body_region = c("head", "torso", "pelvis")
)
bones <- data.frame(
  from_id = c(0, 1),
  to_id = c(1, 2),
  bone_name = c("neck", "spine")
)
hier <- list(Head = "Torso", Torso = "Hip")
sk <- SkeletonModel("mini", kp, bones, hier, "Head")
print(sk)
```

`summarizeGaitParameters` *Summarise gait parameters*

Description

Computes mean, standard deviation, and coefficient of variation for each gait parameter across all strides.

Usage

```
summarizeGaitParameters(gait_params)
```

Arguments

`gait_params`	A `gait_parameters` object.

Value

A data.frame with columns `parameter`, `side`, `mean`, `sd`, and `cv` (coefficient of variation as percentage).

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons.

See Also

`calculateGaitParameters()` for computing gait parameters, `print.gait_parameters()` for formatted display of results.

Examples

```
# See test file for worked examples
```

`symmetryIndex` *Compute bilateral symmetry index*

Description

Calculates a symmetry index between left and right side measurements using standard methods from the biomechanics literature.

Usage

```
symmetryIndex(left, right, method = "robinson")
```

Arguments

`left`	Numeric vector or matrix of left-side values.
`right`	Numeric vector or matrix of right-side values (same dimensions as `left`).
`method`	Character string specifying the symmetry index method. `"robinson"` (default): Robinson (1987) symmetry index $SI = \|L-R\|/(0.5\times(L+R))\times 100$. `"ratio"`: simple ratio L/R.

Details

The Robinson (1987) symmetry index returns 0 for perfect symmetry and larger values for greater asymmetry. It is expressed as a percentage. The ratio method returns 1.0 for perfect symmetry.

Value

For vectors, a numeric vector of symmetry indices. For matrices, row-wise symmetry indices (a numeric vector of length equal to `nrow(left)`).

References

Robinson, R.O., Herzog, W., & Nigg, B.M. (1987). Use of force platform variables to quantify the effects of chiropractic manipulation on gait symmetry. *Journal of Manipulative and Physiological Therapeutics*, 10(4), 172-176.

See Also

`calculateStepSymmetry()` for gait-specific symmetry metrics, `plotSymmetry()` for symmetry visualization, `calculateGaitParameters()` for comprehensive gait analysis.

Examples

```
# Perfect symmetry
symmetryIndex(10, 10)   # returns 0

# Known asymmetry
symmetryIndex(10, 8)    # returns ~22.2%

# Ratio method
symmetryIndex(10, 8, method = "ratio")  # returns 1.25
```

---

synchronizeSignals *Synchronize multiple PhysioExperiment objects*

Description

Resamples a list of PhysioExperiment objects so they all share the same sampling rate and have the same number of time points.

Usage

```
synchronizeSignals(pe_list, target_rate = NULL, method = c("linear", "spline"))
```

Arguments

pe_list	A list of PhysioExperiment objects.
target_rate	Target sampling rate in Hz. If NULL (default), the highest sampling rate among the inputs is used.
method	Interpolation method: "linear" (default) or "spline".

Value

A list of PhysioExperiment objects, all with the same sampling rate and the same number of rows (time points).

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

resampleSignal() for resampling individual PhysioExperiment objects, resampleVector() for resampling raw numeric vectors.

Examples

```
pe1 <- PhysioCore::PhysioExperiment(
  assays = S4Vectors::SimpleList(
    raw = matrix(rnorm(100), ncol = 1)
  ),
  colData = S4Vectors::DataFrame(label = "ch1", type = "marker"),
  samplingRate = 100
)
pe2 <- PhysioCore::PhysioExperiment(
  assays = S4Vectors::SimpleList(
    raw = matrix(rnorm(200), ncol = 1)
  ),
  colData = S4Vectors::DataFrame(label = "ch1", type = "marker"),
  samplingRate = 200
)
synced <- synchronizeSignals(list(pe1, pe2))
```

TaskSchema *Create a Task Schema*

Description

Defines a complete movement task schema including events, phases, normalization method, metrics, and visualization defaults.

Usage

```
TaskSchema(
  task_type,
  task_label = task_type,
  events = list(),
  phases = list(),
  normalization = c("cycle", "phase", "landmark", "dtw", "absolute"),
  norm_length = 101L,
  metrics = character(),
  vis_defaults = list()
)
```

Arguments

task_type	Short identifier for the task type (e.g., "gait", "jump")
task_label	Human-readable label (e.g., "Gait Cycle", "Vertical Jump")
events	List of Event objects defining key timepoints
phases	List of Phase objects defining movement phases
normalization	Normalization method: • "cycle" - Normalize entire movement to 0-100%

- "phase" - Normalize each phase independently to 0-100%
- "landmark" - Align to a specific event
- "dtw" - Dynamic time warping alignment
- "absolute" - Keep original time (no normalization)

`norm_length` Target length after normalization (default 101 for 0-100%)

`metrics` Character vector of recommended metrics for this task

`vis_defaults` List of default visualization parameters:

- xlab - X-axis label
- ylab - Y-axis label
- show_phases - Whether to show phase regions
- show_events - Whether to show event markers
- phase_alpha - Transparency for phase shading
- landmark_event - Event to align to (for landmark normalization)

Value

A TaskSchema object (S3 class)

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

`Event()`, `Phase()`, `getSchema()`, `validateSchema()`

Examples

```
# Simple task schema
my_task <- TaskSchema(
  task_type = "simple",
  task_label = "Simple Movement",
  events = list(
    Event("start", "Start", "threshold",
          list(signal = "velocity", threshold = 0.1, direction = "rising")),
    Event("end", "End", "threshold",
          list(signal = "velocity", threshold = 0.1, direction = "falling"))
  ),
  phases = list(
    Phase("main", "Main Phase", "start", "end")
  ),
  normalization = "cycle"
)
```

`trackMarkers` *Track Markers Across Frames*

Description

Resolves inconsistent marker labelling across frames by establishing frame-to-frame correspondences using the Hungarian algorithm. This is essential for Venus3D data where marker IDs are randomly reassigned each frame.

Usage

```
trackMarkers(
  pe,
  method = "hungarian",
  max_distance = Inf,
  use_prediction = FALSE,
  assay_prefix = "position"
)
```

Arguments

`pe`	A PhysioExperiment with `position_x`, `position_y`, `position_z` assays (frames x markers).
`method`	Assignment method: `"hungarian"` (optimal, requires the `clue` package) or `"greedy"` (fast approximate). Default `"hungarian"`.
`max_distance`	Maximum allowed assignment distance. Assignments exceeding this threshold are set to `NA`. Default `Inf` (no limit).
`use_prediction`	Logical. If `TRUE`, use linear velocity prediction for the reference positions instead of raw previous-frame positions. Improves tracking of fast-moving markers. Default `FALSE`.
`assay_prefix`	Prefix for position assay names. Default `"position"`.

Details

The algorithm:

1. Frame 1 defines the reference labelling.
2. For each subsequent frame, a Euclidean distance cost matrix is computed between reference positions and observed positions.
3. The cost matrix is solved via `clue::solve_LSAP()` (Hungarian algorithm) or a greedy heuristic.
4. If marker counts differ between frames, the cost matrix is padded with dummy entries (cost = 1e12).
5. Assignments exceeding `max_distance` are marked as `NA`.
6. With `use_prediction = TRUE`, reference positions are extrapolated using velocity from the two preceding frames.

Value

A PhysioExperiment where columns consistently correspond to the same physical marker across all frames. The `metadata$tracking` list contains:

assignment Integer matrix (frames x markers) of column indices from the original data used at each frame.

cost Numeric matrix (frames x markers) of assignment costs (Euclidean distances) at each frame.

method Character string indicating the method used.

References

Kuhn HW (1955). "The Hungarian Method for the Assignment Problem." Naval Research Logistics Quarterly, 2(1-2), 83-97.

See Also

`readVenus3D()` for reading Venus3D data, `detectSwaps()` and `correctSwaps()` for post-tracking swap repair.

Examples

```
## Not run:
pe <- readVenus3D("capture.csv")
pe_tracked <- trackMarkers(pe)

# With velocity prediction for fast movements
pe_tracked <- trackMarkers(pe, use_prediction = TRUE, max_distance = 50)

## End(Not run)
```

`validateBenchmarkManifest` *Validate a benchmark manifest*

Description

Checks structural validity and file existence for benchmark inputs.

Usage

```
validateBenchmarkManifest(manifest, data_dir = ".")
```

Arguments

`manifest`	A manifest data.frame or path to a manifest CSV.
`data_dir`	Base directory used to resolve relative file paths.

Value

An object of class "benchmark_manifest_validation" with fields: valid, issues, and manifest.

References

Bland JM, Altman DG (1986). "Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement." Lancet, 327(8476), 307-310.

See Also

benchmarkManifestTemplate() for creating manifest templates, print.benchmark_manifest_validation() for displaying validation results.

Examples

```
m <- benchmarkManifestTemplate(1)
v <- validateBenchmarkManifest(m)
v$valid
```

---

validateSchema *Validate a TaskSchema*

Description

Checks that a TaskSchema is internally consistent.

Usage

```
validateSchema(schema)
```

Arguments

schema	A TaskSchema object

Value

TRUE if valid, otherwise throws an error

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

See Also

TaskSchema(), getSchema(), listSchemas()

`vectorAngle` *Angle between two 3D vectors*

Description

Computes the angle between pairs of 3D vectors. Accepts either single vectors (length-3 numeric) or matrices where each row is a vector (n x 3). Vectorized for efficient computation across time frames.

Usage

```
vectorAngle(v1, v2, degrees = TRUE)
```

Arguments

`v1`	Numeric vector of length 3 or matrix with 3 columns.
`v2`	Numeric vector of length 3 or matrix with 3 columns.
`degrees`	Logical. If `TRUE` (default), return angles in degrees. If `FALSE`, return in radians.

Details

The angle is computed as:

$$\theta = \arccos\left(\frac{v_1 \cdot v_2}{|v_1||v_2|}\right)$$

The dot product is clamped to $[-1, 1]$ before applying `acos` to avoid numerical issues. If either vector has zero magnitude, the result is `NA`.

Value

Numeric vector of angles. Length 1 for vector inputs, or `nrow(v1)` for matrix inputs.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley.

Grood ES, Suntay WJ (1983). "A joint coordinate system for the clinical description of three-dimensional motions: application to the knee." Journal of Biomechanical Engineering, 105(2), 136-144.

See Also

`calculateJointAngles()`, `quaternionToEuler()`, `eulerToQuaternion()`

Examples

```
# 90-degree angle
vectorAngle(c(1, 0, 0), c(0, 1, 0))

# Vectorized across rows
v1 <- matrix(c(1,0,0, 0,1,0), nrow = 2, byrow = TRUE)
v2 <- matrix(c(0,1,0, 0,0,1), nrow = 2, byrow = TRUE)
vectorAngle(v1, v2)
```

---

waveformPCA *PCA for waveform data*

Description

Performs Principal Component Analysis on waveform data, either using extracted features or raw waveforms.

Usage

```
waveformPCA(
  x,
  method = c("features", "raw"),
  features = c("statistical", "shape"),
  n_components = 10,
  scale = TRUE
)
```

Arguments

x	A PhysioExperiment object or matrix.
method	Feature extraction method: "features" or "raw".
features	If method = "features", which features to extract.
n_components	Number of PCs to retain.
scale	Logical; scale features to unit variance.

Value

A list of class "waveform_pca" containing:

scores	PC scores (observations x components)
loadings	PC loadings (features x components)
variance_explained	Variance explained by each PC
cumulative_variance	Cumulative variance
center	Feature means
scale	Feature SDs (if scaled)

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

extractWaveformFeatures(), waveformUMAP(), plotPCAScatter(), plotPCAVariance()

Examples

```
# PCA on gait features
set.seed(123)
data <- matrix(rnorm(1000), nrow = 100, ncol = 10)
pca_result <- waveformPCA(data, method = "features")
plotPCAScatter(pca_result)
```

---

waveformTSNE *t-SNE embedding for waveform data*

Description

Performs t-SNE dimensionality reduction on waveform data. Requires the Rtsne package.

Usage

```
waveformTSNE(
  x,
  perplexity = 30,
  n_components = 2,
  max_iter = 1000,
  features = c("statistical", "shape"),
  use_pca = TRUE,
  n_pca = 30,
  seed = NULL
)
```

Arguments

x	A PhysioExperiment object, matrix, or waveform_pca result.
perplexity	t-SNE perplexity parameter.
n_components	Number of dimensions (usually 2).
max_iter	Maximum iterations.
features	If x is waveform data, features to extract.
use_pca	Logical; pre-reduce with PCA.
n_pca	Number of PCA components.
seed	Random seed.

Value

A list of class "waveform_tsne" containing:

`embedding`	t-SNE coordinates
`n_obs`	Number of observations

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

`waveformPCA()`, `waveformUMAP()`, `extractWaveformFeatures()`

`waveformUMAP` *UMAP embedding for waveform data*

Description

Performs UMAP dimensionality reduction on waveform data for visualization. Requires the uwot package.

Usage

```
waveformUMAP(
  x,
  n_neighbors = 15,
  n_components = 2,
  min_dist = 0.1,
  metric = "euclidean",
  features = c("statistical", "shape"),
  use_pca = TRUE,
  n_pca = 30,
  seed = NULL
)
```

Arguments

`x`	A PhysioExperiment object, matrix, or waveform_pca result.
`n_neighbors`	Number of neighbors for UMAP.
`n_components`	Number of UMAP dimensions (usually 2).
`min_dist`	Minimum distance parameter.
`metric`	Distance metric: "euclidean", "cosine", "manhattan".
`features`	If x is waveform data, features to extract.

use_pca	Logical; pre-reduce with PCA (recommended for high-dim).
n_pca	Number of PCA components to use as input.
seed	Random seed for reproducibility.

Value

A list of class "waveform_umap" containing:

embedding	UMAP coordinates (observations x n_components)
n_obs	Number of observations
params	UMAP parameters used

References

van der Maaten L, Hinton G (2008). "Visualizing Data using t-SNE." Journal of Machine Learning Research, 9, 2579-2605.

See Also

waveformPCA(), waveformTSNE(), plotUMAP(), extractWaveformFeatures()

Examples

```
## Not run:
# UMAP on gait data
data <- matrix(rnorm(1000), nrow = 100, ncol = 10)
umap_result <- waveformUMAP(data, n_neighbors = 15)
plotUMAP(umap_result)

## End(Not run)
```

---

writeBenchmarkManifest

Write a benchmark manifest template to CSV

Description

Write a benchmark manifest template to CSV

Usage

```
writeBenchmarkManifest(path, n = 1L, overwrite = FALSE)
```

Arguments

path	Output CSV path.
n	Number of template rows.
overwrite	Logical; overwrite existing file if TRUE.

Value

Invisibly returns `path`.

References

Bland JM, Altman DG (1986). "Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement." Lancet, 327(8476), 307-310.

See Also

`benchmarkManifestTemplate()` for creating the manifest data.frame, `validateBenchmarkManifest()` for validating manifest structure.

Examples

```
## Not run:
writeBenchmarkManifest("benchmark_manifest.csv", n = 3)

## End(Not run)
```

Index

∗ **datasets**
schema_balance, 156
schema_cutting, 157
schema_cycling, 157
schema_gait, 158
schema_jump, 159
schema_running, 159
schema_throw, 160

alignEMGtoMoCap, 6
alignEMGtoMoCap(), *78*
analyzeForcePlate, 7
analyzeForcePlate(), *9*, *19*, *29*, *32*, *44*, *64*, *118*
analyzeForcePlatePE, 8
analyzeForcePlatePE(), *8*, *44*, *118, 119*
approx, *61, 62*, *152*
assessMoCapReadiness, 9
assessMoCapReadiness(), *41*, *122*, *128*, *137*

batch_analyze_opensim, 10
batch_analyze_opensim(), *37*, *153*
batchNormalize, 12
batchNormalize(), *27*, *85*, *88*
benchmarkAgreement, 13
benchmarkAgreement(), *15*, *39*, *115*, *154*
benchmarkManifestTemplate, 14
benchmarkManifestTemplate(), *38*, *172*, *178*
blandAltman, 14
blandAltman(), *13*, *76*
butterworthFilter, 16
butterworthFilter(), *64*, *66*, *84*, *155*

calculateCOM, 17
calculateCOM(), *19*, *23*, *161*
calculateCOP, 18
calculateCOP(), *8*
calculateGaitParameters, 19
calculateGaitParameters(), *24*, *98*, *102*, *120*, *165, 166*
calculateJointAngles, 21
calculateJointAngles(), *57*, *127*, *173*
calculateSegmentCOM, 22
calculateSegmentCOM(), *18*, *161*
calculateStepSymmetry, 23
calculateStepSymmetry(), *20*, *111*, *166*
calibrateIMU, 24
calibrateIMU(), *54*, *150*
cohensD, 26
cohensD(), *56*, *95*
combineTrials, 27
combineTrials(), *12*
computeAcceleration, 28
computeAcceleration(), *30*, *34, 35*, *47*
computeImpulse, 29
computeImpulse(), *32*
computeJerk, 30
computeJerk(), *28*
computeJointPower, 31
computeJointPower(), *79, 80*
computeLoadingRate, 31
computeLoadingRate(), *8*, *29*
computeRMSEnvelope, 32
computeRMSEnvelope(), *86*, *125*, *147*
computeSpeed, 33
computeSpeed(), *35*
computeVelocity, 34
computeVelocity(), *28*, *30*, *34*, *47*, *103*
correctSwaps, 35
correctSwaps(), *45*, *171*
create_schema_from_opensim, 36
create_schema_from_opensim(), *11*, *153*
createBenchmarkExample, 37
createBenchmarkExample(), *154*

defaultBenchmarkThresholds, 39
defaultBenchmarkThresholds(), *13*
define_skeleton, 39

define_skeleton(), *68, 69*, *135*, *142*, *164*
demoMoCapData, 40
demoMoCapData(), *10*, *128*
detectEvents, 42
detectEvents(), *37*, *58*, *82*, *162*
detectForcePlateContacts, 43
detectGaps, 44, *150*
detectGaps(), *61–63*, *150*
detectSwaps, 45
detectSwaps(), *35*, *171*
differentiate, 46
differentiate(), *28*, *30*, *35*, *155*
dtwAverage, 47
dtwAverage(), *49–52*
dtwClustering, 48
dtwClustering(), *48*, *50, 51*, *94*
dtwDistance, 49
dtwDistance(), *48, 49*, *51, 52*, *94*
dtwDistanceMatrix, 51
dtwDistanceMatrix(), *48–50*
dtwWarp, 52
dtwWarp(), *50*, *94*

estimateOrientation, 53
estimateOrientation(), *25*, *150*
estimateSegmentInertia, 54
estimateSegmentInertia(), *79, 80*, *161*
etaSquared, 55
etaSquared(), *27*, *95*
eulerToQuaternion, 56
eulerToQuaternion(), *22*, *127*, *173*
Event, 57
Event(), *88*, *169*
extractPhase, 59
extractPhase(), *73*, *75*, *162*
extractWaveformFeatures, 60
extractWaveformFeatures(), *175–177*

fillGaps, 61
fillGaps(), *44*, *62, 63*, *150*
fillGapsLinear, 62
fillGapsLinear(), *44*, *61*, *63*, *150*
fillGapsSpline, 63
fillGapsSpline(), *61, 62*
filterGRF, 64
filterGRF(), *8*
filterSignals, 65
filterSignals(), *16*
fPCA, 66
fPCA(), *96*, *120*, *146*, *148*

get_bone_connections, 67
get_bone_connections(), *40*, *69*, *164*
get_limb_pairs, 68
get_limb_pairs(), *40*, *69*
get_segment_lengths, 69
get_segment_lengths(), *68, 69*, *164*
getEvent, 70
getEvent(), *71, 72*
getEventNames, 71
getEventNames(), *70*, *74*
getPhase, 71
getPhase(), *70*, *72*, *74*
getPhaseColors, 72
getPhaseColors(), *72*, *74*
getPhaseData, 73
getPhaseData(), *59*, *75*
getPhaseNames, 73
getPhaseNames(), *71, 72*
getSchema, 74
getSchema(), *81*, *158*, *169*, *172*

hasValidPhases, 75
hasValidPhases(), *73*

icc, 75
icc(), *15*, *83*, *163*
integrateEMGMoCap, 77
integrateEMGMoCap(), *6*, *125*
inverseDynamics2D, 78
inverseDynamics2D(), *31*, *55*, *80*
inverseDynamics3D, 79
inverseDynamics3D(), *31*, *55*, *79*

listSchemas, 81
listSchemas(), *74*, *172*

manualEvents, 82
manualEvents(), *43*
mdc, 83
mdc(), *76*, *163*
movingAverage, 84
movingAverage(), *66*, *155*

normalizedTimeAxis, 85
normalizedTimeAxis(), *12*, *88*
normalizeEMG, 86
normalizeEMG(), *33*, *125*
normalizeMovement, 87

normalizeMovement(), *12*, *27*, *52*, *85*, *162*

Phase, 88
Phase(), *58*, *169*
phaseDurations, 89
phaseDurations(), *90, 91*
phaseRatios, 90
phaseRatios(), *89*, *91*
phaseTiming, 90
phaseTiming(), *59*, *89, 90*, *162*
plotCorrelationMatrix, 91
plotCorrelationMatrix(), *95*
plotCycle, 92
plotCycle(), *99, 100*, *102*
plotDTW, 94
plotEffectSizeForest, 95
plotEffectSizeForest(), *27*, *56*, *92*
plotFPCA, 96
plotFPCA(), *67*, *120*, *146*
plotGaitCycle, 97
plotGaitCycle(), *20*, *103*, *110*, *114*, *148*
plotGroupComparison, 98
plotGroupComparison(), *93*, *100*
plotMultiPanel, 99
plotMultiPanel(), *93*
plotPCAScatter, 100
plotPCAScatter(), *101*, *113*, *175*
plotPCAVariance, 101
plotPCAVariance(), *100*, *175*
plotPhaseDurations, 101
plotPhaseDurations(), *93*
plotPhasePortrait, 102
plotPhasePortrait(), *112*
plotSkeleton, 103
plotSkeleton(), *106, 107*, *109*, *112*, *126*
plotSkeleton3D, 105
plotSkeleton3D(), *104*, *126*
plotSkeletonOverlay, 106
plotSkeletonOverlay(), *104*, *109*
plotSkeletonSequence, 108
plotSkeletonSequence(), *104*, *107*
plotSpaghetti, 109
plotSpaghetti(), *98*, *114*
plotSymmetry, 110
plotSymmetry(), *24*, *114*, *166*
plotTrajectory, 111
plotUMAP, 112
plotUMAP(), *100, 101*, *177*
plotWaveformComparison, 113
plotWaveformComparison(), *92*, *98, 99*, *110, 111*
print.ASFSkeleton, 114
print.ASFSkeleton(), *130*
print.benchmark_agreement, 115
print.benchmark_manifest_validation, 116
print.benchmark_manifest_validation(), *172*
print.benchmark_suite, 116
print.benchmark_suite(), *154*
print.detected_events, 117
print.dtw_clustering, 117
print.dtw_result, 118
print.forceplate_analysis, 118
print.forceplate_analysis(), *9*, *119*
print.forceplate_analysis_multi, 119
print.fpca_result, 119
print.gait_parameters, 120
print.gait_parameters(), *165*
print.mocap_quickstart, 121
print.mocap_quickstart(), *128*
print.mocap_readiness, 121
print.mocap_readiness(), *10*
print.multi_trial_phases, 122
print.segmented_phases, 122
print.SkeletonModel, 123
print.waveform_pca, 123
print.waveform_umap, 124
processEMG, 124
processEMG(), *33*, *78*, *86*, *147*
projectTo2D, 125
projectTo2D(), *104*, *106*

quaternionToEuler, 126
quaternionToEuler(), *22*, *54*, *57*, *150*, *173*
quickStartMoCap, 127
quickStartMoCap(), *10*, *41*, *121*

readAMC, 129
readAMC(), *115*, *130*, *137, 138*
readASF, 130
readASF(), *115*, *129*, *137, 138*
readBVH, 131
readBVH(), *132*
readC3D, 132
readC3D(), *131*, *136*, *141*, *144*
readDeepLabCut, 133
readDeepLabCut(), *135*, *142*

readMediaPipe, 134
readMediaPipe(), *134*, *142*
readMoCapAuto, 136
readMoCapAuto(), *138*, *146*
readMoCapCSV, 137
readMoCapCSV(), *129*, *137*, *146*
readMOT, 139, *140, 141*
readMOT(), *141*, *143, 144*
readOpenCap, 140
readOpenCap(), *134*, *139*, *144*
readOpenPose, 141
readOpenPose(), *131, 132*, *134, 135*
readOpenSimOutputs, 143
readSTO, 143
readSTO(), *139*, *143, 144*
readTRC, *140, 141*, 144
readTRC(), *131, 132*, *139*, *141*, *143, 144*
readVenus3D, 145
readVenus3D(), *171*
reconstructFPCA, 146
reconstructFPCA(), *67*, *96*, *120*
rectifyEMG, 147
rectifyEMG(), *33*, *86*, *125*
registerCurves, 148
registerCurves(), *67*
removeGravity, 149
removeGravity(), *25*, *54*
reportGaps, 150
reportGaps(), *44*, *61*
resampleSignal, 151
resampleSignal(), *6*, *152*, *167*
resampleVector, 152
resampleVector(), *151*, *167*
run_opensim_toolchain, 153
runBenchmarkSuite, 154
runBenchmarkSuite(), *13*, *38, 39*, *117*

savgolFilter, 155
savgolFilter(), *16*, *47*, *66*, *84*
schema_balance, 156
schema_cutting, *156*, 157, *160*
schema_cycling, 157
schema_gait, *156, 157*, 158, *158–160*
schema_jump, 159, *160*
schema_running, *157, 158*, 159
schema_throw, *159*, 160
segmentParameters, 161
segmentParameters(), *17, 18*, *23*, *55*
segmentPhases, 162
segmentPhases(), *27*, *43*, *59*, *73*, *75*, *82*, *85*, *88–91*
sem, 163
sem(), *76*, *83*
signal::butter(), *65*
signal::filtfilt(), *65*
SkeletonModel, 164
SkeletonModel(), *40*, *68, 69*
smooth.spline, *61*, *63*
spline, *152*
stats::approx(), *6*
stats::filter(), *84*
summarizeGaitParameters, 165
summarizeGaitParameters(), *20*, *120*
symmetryIndex, 166
symmetryIndex(), *18*, *24*, *111*
synchronizeSignals, 167
synchronizeSignals(), *78*, *151, 152*

TaskSchema, 168
TaskSchema(), *43*, *58*, *70–72*, *74*, *81, 82*, *88*, *156–160*, *172*
trackMarkers, 170
trackMarkers(), *35*, *45*, *145, 146*

validateBenchmarkManifest, 171
validateBenchmarkManifest(), *14*, *116*, *178*
validateSchema, 172
validateSchema(), *74*, *81*, *169*
vectorAngle, 173
vectorAngle(), *22*, *57*, *127*

waveformPCA, 174
waveformPCA(), *60*, *100, 101*, *113*, *176, 177*
waveformTSNE, 175
waveformTSNE(), *60*, *177*
waveformUMAP, 176
waveformUMAP(), *60*, *113*, *175, 176*
writeBenchmarkManifest, 177
writeBenchmarkManifest(), *14*