Package: PhysioMoCap (via r-universe)

October 3, 2026

Title Motion Capture and Biomechanics Analysis for PhysioExperiment Objects

Version 0.7.5

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), PhysioExperiment (>= 2.0.0)

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

Suggests PhysioCore, PhysioIO, PhysioOpenSim, PhysioGaitNorm, signal, uwot, Rtsne, c3dr, clue, rhdf5, httr, withr, quadprog, knitr, rmarkdown, testthat (>= 3.2.0)

VignetteBuilder knitr

License MIT + file LICENSE

URL https://github.com/x-biosignal/PhysioMoCap, https://x-biosignal.r-universe.dev/PhysioMoCap, https://x-biosignal.github.io/PhysioMoCap/

BugReports https://github.com/x-biosignal/PhysioMoCap/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Collate 'class-TaskSchema.R' 'utils-provenance.R' 'schemas-prebuilt.R'
'mocap-skeleton.R' 'mocap-angles.R' 'anatomical-frames.R'
'mocap-gaps.R' 'mocap-imu.R' 'imu-gait.R' 'mocap-tracking.R'
'io-openpose.R' 'io-opensim.R' 'pose-fix.R' 'io-c3d.R'
'io-bvh.R' 'io-deeplabcut.R' 'io-mediapipe.R' 'io-opencap.R'
'ops-events.R' 'events-zeni.R' 'ops-phases.R' 'ops-normalize.R'
'ops-dtw.R' 'ops-dimred.R' 'stats-fda.R'
'opensim-integration.R' 'static-optimization.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'
'vis-clinical.R' 'stats-gait.R' 'kinetics-forceplate.R'
'kinetics-inverse-dynamics.R' 'kinetics-rne.R' 'gait-indices.R'
'mocap-emg.R' 'dynamometry.R' 'user-onboarding.R'
'benchmark-validation.R' 'quality-smoothness.R' 'reaching.R'
'stability-nonlinear.R' 'gait-pathology.R'
'pipelines-population.R' 'clinical-mobility-tests.R'
'posturography.R' 'plantar-pressure.R' 'vis-pressure.R'
'adl-reach.R' 'coordination.R' 'stats-reliability.R'
'stats-circular.R' 'motor-variability-ucm.R'
'kinetics-forward-dynamics.R' 'motor-primitives.R'
'stability-orbital-rqa.R' 'imu-fusion.R' 'inverse-kinematics.R'
'movement-tremor.R' 'ataxia.R' 'spasticity-tardieu.R' 'zzz.R'

RoxygenNote 7.3.3

Config/pak/sysreqs zlib1g-dev

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

Date/Publication 2026-10-03 13:11:45 UTC

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

RemoteRef HEAD

RemoteSha 6d95583987dc285f292a8a8023f455d873ce94b3

Contents

adlReachTask 8
alignEMGtoMoCap 9
analyzeForcePlate 11
analyzeForcePlatePE 12
approximateEntropy 13
assessMoCapReadiness 14
batch_analyze_opensim 15
batchNormalize 17
benchmarkAgreement 18
benchmarkManifestTemplate 19
butterworthFilter 19
c3dEventTable 21
calculateCOM 22

calculateCOP . . . 24
calculateGaitParameters . . . 25
calculateJointAngles . . . 27
calculateSegmentCOM . . . 29
calculateStepSymmetry . . . 30
calibrateIMU . . . 31
classifyGaitPatterns . . . 32
combineTrials . . . 33
computeAcceleration . . . 34
computeImpulse . . . 35
computeJerk . . . 36
computeJointPower . . . 37
computeLoadingRate . . . 38
computeRMSEnvelope . . . 39
computeSpeed . . . 40
computeVelocity . . . 41
continuousRelativePhase . . . 42
contractionWork . . . 43
coordinationVariability . . . 44
copFromPressure . . . 45
correctSwaps . . . 46
create_schema_from_opensim . . . 47
createBenchmarkExample . . . 48
defaultBenchmarkThresholds . . . 49
define_skeleton . . . 50
demoMoCapData . . . 51
detectCircumduction . . . 52
detectEvents . . . 53
detectEventsZeni . . . 54
detectForcePlateContacts . . . 56
detectGaps . . . 57
detectPushOffDeficit . . . 58
detectStanceZUPT . . . 59
detectStiffKnee . . . 60
detectSwaps . . . 62
detectTrendelenburg . . . 63
differentiate . . . 64
dimensionlessJerk . . . 65
dmpFit . . . 65
dmpGenerate . . . 66
downloadOpenCapModel . . . 67
dtwAverage . . . 68
dtwClustering . . . 69
dtwDistance . . . 71
dtwDistanceMatrix . . . 72
dtwWarp . . . 73
endpointError . . . 74
estimateOrientation . . . 74

estimateSegmentInertia . . . 76
eulerToQuaternion . . . 77
Event . . . 78
extractPhase . . . 79
extractWaveformFeatures . . . 80
fillGaps . . . 82
fillGapsLinear . . . 83
fillGapsSpline . . . 83
filterGRF . . . 84
filterSignals . . . 85
floquetMultipliers . . . 87
footImuGait . . . 88
forwardDynamics . . . 89
forwardKinematics2D . . . 90
gaitAtaxiaIndex . . . 91
gaitDeviationIndex . . . 92
gaitPatternLibrary . . . 93
gaitProfileScore . . . 93
gaitVariableScore . . . 94
gdiBasis . . . 95
get_bone_connections . . . 96
get_limb_pairs . . . 97
get_segment_lengths . . . 97
getEvent . . . 98
getEventNames . . . 99
getPhase . . . 100
getPhaseColors . . . 100
getPhaseData . . . 101
getPhaseNames . . . 102
getSchema . . . 103
goalEquivalentManifold . . . 103
grfLandmarks . . . 104
groodSuntayAngles . . . 105
hamstringQuadRatio . . . 107
harmonicRatio . . . 108
hasValidPhases . . . 109
instrumented10mWT . . . 110
instrumentedTUG . . . 111
integrateEMGMoCap . . . 112
inverseDynamics2D . . . 114
inverseDynamics3D . . . 116
inverseDynamicsRNE . . . 118
inverseKinematicsMarkers . . . 119
jointCoordinateSystem . . . 120
jointWork . . . 121
kalmanOrientation . . . 122
labelPowerBursts . . . 124
ldlj . . . 125

limbAtaxiaIndex . . . 126
limitsOfStability . . . 127
listSchemas . . . 128
localDynamicStability . . . 129
mahonyAHRS . . . 130
manualEvents . . . 131
maxLyapunovExponent . . . 132
mCTSIB . . . 133
movementAnalysisProfile . . . 134
movementSmoothness . . . 135
movementTime . . . 136
movementUnits . . . 136
movingAverage . . . 137
nhptDexterity . . . 138
normalizedTimeAxis . . . 139
normalizeEMG . . . 140
normalizeMovement . . . 141
pathStraightness . . . 142
peakPressure . . . 143
peakTorque . . . 143
peakVelocity . . . 145
Phase . . . 145
phaseDurations . . . 146
phaseRatios . . . 147
phaseTiming . . . 148
pipelineACLrts . . . 149
pipelineAmputee . . . 150
pipelineCPgait . . . 151
pipelinePDfog . . . 152
pipelineStrokePropulsion . . . 153
planarLimbModel . . . 154
plotBlandAltman . . . 155
plotCorrelationMatrix . . . 156
plotCycle . . . 157
plotDTW . . . 159
plotEffectSizeForest . . . 160
plotFPCA . . . 161
plotGaitCycle . . . 162
plotGroupComparison . . . 163
plotMAP . . . 164
plotMultiPanel . . . 165
plotPCAScatter . . . 166
plotPCAVariance . . . 167
plotPhaseDurations . . . 168
plotPhasePortrait . . . 169
plotPressureMap . . . 170
plotSkeleton . . . 171
plotSkeleton3D . . . 172

plotSkeletonOverlay . . . 174
plotSkeletonSequence . . . 175
plotSpaghetti . . . 177
plotSymmetry . . . 178
plotTrajectory . . . 179
plotTUG . . . 180
plotUMAP . . . 181
plotWaveformComparison . . . 182
poseFix . . . 183
poseLengthCorrect . . . 185
predictiveSimulation . . . 186
pressureAsymmetry . . . 187
pressureMovie . . . 188
pressureTimeIntegral . . . 189
print.ASFSkeleton . . . 190
print.benchmark_agreement . . . 191
print.benchmark_manifest_validation . . . 192
print.benchmark_suite . . . 193
print.detected_events . . . 194
print.dtw_clustering . . . 194
print.dtw_result . . . 195
print.forceplate_analysis . . . 195
print.forceplate_analysis_multi . . . 196
print.gait_parameters . . . 197
print.mocap_quickstart . . . 198
print.mocap_readiness . . . 199
print.multi_trial_phases . . . 200
print.segmented_phases . . . 200
print.SkeletonModel . . . 201
print.waveform_pca . . . 202
print.waveform_umap . . . 202
processEMG . . . 203
projectTo2D . . . 204
promCondition . . . 205
promFit . . . 205
quaternionToEuler . . . 206
quickStartMoCap . . . 208
rateOfForceDevelopment . . . 209
reachingKinematics . . . 210
readAMC . . . 212
readASF . . . 213
readBVH . . . 214
readC3D . . . 215
readDeepLabCut . . . 216
readGaitRec . . . 217
readMediaPipe . . . 219
readMoCapAuto . . . 220
readMoCapCSV . . . 221

readMOT . . . 223
readOpenCap . . . 224
readOpenPose . . . 226
readOpenSimOutputs . . . 227
readPressureFrame . . . 228
readSTO . . . 229
readTRC . . . 230
readVenus3D . . . 231
rectifyEMG . . . 232
recurrenceQuantification . . . 233
reflexThreshold . . . 234
regionalLoading . . . 235
removeGravity . . . 236
reportGaps . . . 237
resampleSignal . . . 238
resampleVector . . . 239
run_opensim_toolchain . . . 240
runBenchmarkSuite . . . 241
runCMC . . . 242
runOpenSimFromMarkers . . . 243
runOpenSimFromOpenCap . . . 245
runRRA . . . 246
runStaticOptimization . . . 247
sampleEntropy . . . 248
savgolFilter . . . 249
scaleBodyModel . . . 250
scalePlanarModel . . . 251
scaleSegmentModel . . . 252
schema_balance . . . 253
schema_cutting . . . 253
schema_cycling . . . 254
schema_gait . . . 255
schema_jump . . . 255
schema_running . . . 256
schema_throw . . . 257
segmentCohortCycles . . . 257
segmentMultimodalCycles . . . 259
segmentParameters . . . 260
segmentPhases . . . 261
sensoryOrganizationTest . . . 262
SkeletonModel . . . 263
sparc . . . 264
stabilogramDiffusion . . . 265
staticOptimizationR . . . 266
strapdownIntegrate . . . 268
summarizeCycleFeatures . . . 269
summarizeGaitParameters . . . 270
swayMetrics . . . 270

symmetryIndex . . . 272
synchronizeSignals . . . 273
tardieuScore . . . 274
tardieuStretch . . . 275
TaskSchema . . . 276
timeDelayEmbed . . . 278
timeToPeakTorque . . . 279
timeToPeakVelocity . . . 280
toleranceNoiseCovariation . . . 281
trackMarkers . . . 282
tremorAmplitude . . . 283
tremorMetrics . . . 284
tremorSpectrum . . . 285
trunkCompensation . . . 286
uncontrolledManifold . . . 287
validateBenchmarkManifest . . . 288
validateSchema . . . 289
vectorAngle . . . 289
vectorCoding . . . 291
waveformPCA . . . 292
waveformTSNE . . . 293
waveformUMAP . . . 294
writeBenchmarkManifest . . . 296
writeC3D . . . 297
writeMOT . . . 298
writeTRC . . . 299

Index **300**

`adlReachTask` *Assess an upper-limb ADL reach from its speed profile*

Description

Computes the standard reach-to-grasp kinematics for one ADL arm transport and labels it with the ADL task and its ICF code. Works directly on the end-effector (hand) speed profile, so it needs no full marker set; derive the speed from a trajectory with `computeSpeed()` or pass a hand-sensor speed.

Usage

```
adlReachTask(
  speed,
  sampling_rate,
  task = c("reaching", "drinking", "feeding", "dressing", "grooming"),
  onset_threshold = 0.05
)
```

Arguments

speed	Numeric hand/end-effector speed profile (one reach).
sampling_rate	Sampling rate in Hz.
task	ADL task: "reaching" (default, generic hand-arm use, d445), "drinking" (d560), "feeding" (eating, d550), "dressing" (d540) or "grooming" (d520).
onset_threshold	Movement-onset threshold (fraction of peak speed).

Value

an adl_reach_task list: task, icf_code, and the kinematics movement_time, peak_velocity, time_to_peak_frac, movement_units (submovements; 1 = a single smooth transport), sparc and ldlj (smoothness; less negative = smoother).

References

Rohrer B, et al. (2002) J Neurosci 22:8297-8304 (movement units); Balasubramanian S, et al. (2015) IEEE TBME 62:2137-2147 (SPARC).

See Also

reachingKinematics(), movementUnits(), sparc()

Examples

```
# a single smooth (minimum-jerk) reach: one movement unit
fs <- 100; tt <- seq(0, 1, 1 / fs)
v <- 30 * (tt^2) * (1 - tt)^2                # bell-shaped speed
adlReachTask(v, fs, task = "drinking")$movement_units
```

---

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",
  emg_start_time = 0,
  mocap_start_time = 0,
  outside = c("extend", "NA", "error")
)
```

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()`.
`emg_start_time, mocap_start_time`	Finite scalar start times in seconds on the same reference clock. Defaults assume both recordings start at zero. These are known offsets; no synchronization or clock drift is estimated.
`outside`	Handling of target times outside each channel's finite-data support: `"extend"` repeats endpoints (compatibility default), `"NA"` leaves unavailable times missing, and `"error"` rejects the alignment. Fewer than two finite observations produce missing output, or an error under `"error"`. Internal gaps are still interpolated; this policy checks support boundaries, not internal gap duration. No anti-alias filter is added. Endpoint differences within floating-point roundoff are snapped to the endpoint (tolerance capped at one millionth of the shorter sample interval).

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

```
n <- 500
z <- matrix(0, n, 1, dimnames = list(NULL, "fp1"))
fz <- matrix(c(rep(0, 100), rep(700, 300), rep(0, 100)), n, 1,
             dimnames = list(NULL, "fp1"))
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    force_x = z, force_y = z, force_z = fz,
    moment_x = matrix(50, n, 1, dimnames = list(NULL, "fp1")),
    moment_y = matrix(-100, n, 1, dimnames = list(NULL, "fp1")),
    moment_z = z),
  colData = S4Vectors::DataFrame(label = "fp1", type = "forceplate"),
  samplingRate = 1000)
out <- analyzeForcePlatePE(pe, threshold = 20, cutoff = 20,
  filter_method = "moving_average")
out
```

---

approximateEntropy *Approximate entropy (Pincus)*

Description

The regularity of a time series: the (negative) log-likelihood that patterns close for `m` samples stay close for `m + 1`. Lower = more regular/predictable, higher = more irregular. The ApEn complement to the package's sample entropy (ApEn includes self-matches and is more biased for short series, but is the classical measure).

Usage

```
approximateEntropy(x, m = 2L, r = NULL)
```

Arguments

`x`	Numeric time series.
`m`	Pattern length (default 2).
`r`	Tolerance; if `NULL`, `0.2 * sd(x)`.

Value

the approximate entropy (scalar).

References

Pincus SM (1991) PNAS 88:2297-2301.

See Also

recurrenceQuantification(), sampleEntropy()

Examples

```
t <- seq(0, 20 * pi, length.out = 500)
approximateEntropy(sin(t))                          # low (regular)
set.seed(1); approximateEntropy(rnorm(500))         # high (irregular)
```

---

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

Details

Requires a native OpenSim installation (`SystemRequirements: OpenSim`) and setup/model files, so the example cannot run offline and is not executed.

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()`

Examples

```
set.seed(1)
trials <- list(matrix(rnorm(200), 100, 2), matrix(rnorm(180), 90, 2))
norm <- batchNormalize(trials, method = "cycle")
dim(norm[[1]])
```

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)
```

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

```
if (requireNamespace("signal", quietly = TRUE)) {
  x <- sin(2 * pi * 2 * seq(0, 1, length.out = 120)) + rnorm(120, 0, 0.1)
  xf <- butterworthFilter(x, cutoff = 6, sampling_rate = 120)
  length(xf)
}
```

c3dEventTable *Extract C3D Parameter Events on the Recording Clock*

Description

Converts the EVENT parameter group retained by readC3D() into a table of seconds relative to the first stored point sample. The event clock must be declared explicitly. Header events are not read by this function.

Usage

```
c3dEventTable(
  x,
  event_time_reference,
  first_frame = NULL,
  snap = c("none", "point"),
  tolerance = 1e-05,
  outside = c("error", "keep")
)
```

Arguments

x	A PhysioExperiment with metadata(x)$c3d_parameters.
event_time_reference	Either "trial" (C3D frame 1 is time zero) or "recording" (first stored sample is time zero). Required; not inferred.
first_frame	Optional positive, one-based original first point frame, for trial-referenced events. Otherwise read from TRIAL:ACTUAL_START_FIELD. An explicit value overrides the stored parameter and is reported in output.
snap	"none" preserves continuous event times. "point" explicitly requests rounding to the nearest point-sample grid, for frame annotations.
tolerance	Maximum absolute rounding residual in seconds when snap="point". Must be nonnegative and less than half a point period.
outside	"error" rejects events outside the stored point sample times; "keep" retains them with in_recording=FALSE.

Details

Point rate must match samplingRate(x). A trial origin is never silently replaced with zero. ACTUAL_START_FIELD comprises two unsigned 16-bit words, low word first; signed 16-bit storage is decoded as unsigned. Snapping is opt-in because a continuous event need not lie on a point frame. This adapter does not change the signals or their local sample grid. Applying it after cropping requires updating the first frame to match the stored samples.

Operates on a parsed C3D object from the optional c3dr package, built from a binary .c3d file, so the example cannot run offline and is not executed.

Value

A data frame in original event order with `event_index`, `label`, `context`, `raw_minutes`, `raw_seconds`, `raw_time`, `time`, `snap_residual_s`, `recording_origin`, `first_frame`, `event_time_reference`, `snap`, and `in_recording`. `time` is relative to the recording start; `raw_time` is the declared event clock. Residual is raw minus snapped time on that clock, or zero when snapping is disabled. No-event files return a typed empty table. No sorting, event detection, or gait-cycle inference is performed. Event accuracy and the declared clock remain caller concerns.

References

C3D.org, EVENT:TIMES: https://www.c3d.org/HTML/Documents/eventtimes1.htm

See Also

`readC3D()`, `segmentCohortCycles()`

Examples

```
## Not run:
# `x` is a C3D object read via the c3dr package from a .c3d file.
c3dEventTable(x, event_time_reference = "trial_start")

## 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

```
n <- 50
mk <- function(v) { m <- matrix(v, n, 3); colnames(m) <- c("A", "B", "C"); m }
pe <- PhysioExperiment(
  assays = S4Vectors::SimpleList(position_x = mk(1), position_y = mk(2),
                                 position_z = mk(0)),
  colData = S4Vectors::DataFrame(label = c("A", "B", "C"),
                                 type = rep("marker", 3)),
  samplingRate = 100)
bsip <- data.frame(segment = c("s1", "s2"), mass_fraction = c(50, 50),
  com_proximal_fraction = c(0.5, 0.5),
  proximal_marker = c("A", "B"), distal_marker = c("B", "C"),
  stringsAsFactors = FALSE)
pe_com <- calculateCOM(pe, body_mass = 70, bsip = bsip,
```

```
  marker_map = list(s1 = c("A", "B"), s2 = c("B", "C")))
SummarizedExperiment::assayNames(pe_com)
```

`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", "ISB", "groodsuntay"),
  degrees = TRUE,
  signed = FALSE,
  plane_normal = NULL
)
```

Arguments

`pe`	A `PhysioExperiment` object with position assays (`position_x`, `position_y`, and optionally `position_z`).
`joints`	A named list of joint definitions. For `"3point"`, each element is a list with components `proximal`, `joint`, and `distal` (marker column names). For `"groodsuntay"` / `"ISB"`, each element is a list with `proximal` and `distal` *segment* specs, each itself a list of `proximal`/`distal`/`lateral` marker names (see `jointCoordinateSystem()`).
`convention`	Angle convention: `"3point"` (unsigned angle at the vertex, the default) or `"groodsuntay"` / `"ISB"` (signed 3-DOF flexion, ab/adduction and internal/external rotation from anatomical segment frames).
`degrees`	Logical. If `TRUE` (default), return angles in degrees. If `FALSE`, return angles in radians.
`signed`	Logical, `"3point"` only. If `FALSE` (default) the vertex angle is the unsigned $[0, 180]$ included angle. If `TRUE` the angle carries the sign of the rotation from the proximal to the distal vector about `plane_normal`, which separates flexion from hyperextension (see Details).
`plane_normal`	The axis the signed angle is measured about, used only when `signed = TRUE`. Either `NULL` (the default: the global z axis for 2D data, or a best-fit plane normal derived from the data for 3D data), a length-3 numeric vector, an `n_frames x 3` matrix of per-frame normals, or a named list giving any of those per joint.

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 with a "joint_angles" assay. For "3point" this is n_frames x n_joints (one angle per joint: unsigned in $[0, 180]$, or signed in $(-180, 180]$ when signed = TRUE). For "groodsuntay" / "ISB" it is n_frames x (3 * n_joints): three columns per joint (<joint>_flexion, <joint>_abduction, <joint>_rotation) with colData carrying the joint and axis labels.

Signed vertex angles

The unsigned vertex angle cannot tell flexion from hyperextension: a knee flexed 10 degrees and a knee hyperextended 10 degrees both give an included angle of 170 degrees, because $\arccos$ discards which side of the proximal segment the distal segment lies on. With signed = TRUE the angle is instead

$$\theta = \operatorname{atan2}\left((v_1' \times v_2') \cdot \hat{n},\ v_1' \cdot v_2'\right)$$

where $\hat{n}$ is the unit plane_normal and v_i' is v_i with its $\hat{n}$ component removed. The magnitude is the same included angle as before whenever the three points lie in the plane; the sign is positive when the rotation from v_1 to v_2 follows the right-hand rule about $\hat{n}$, and flips when the distal segment crosses to the other side of the proximal segment. Choose (or negate) plane_normal so that this direction is flexion, giving the clinical flexion(+) / extension(-) convention; the corresponding deviation from the straight (anatomically neutral) position is 180 - abs(theta).

Because the range is $(-180, 180]$, the signed angle wraps at exactly the straight configuration - the pose whose two sides the sign is there to tell apart. A joint that moves through full extension therefore steps from +180 to -180, and differentiating that series directly gives a spurious spike. Work with 180 - abs(theta) (which is continuous through the neutral pose) or unwrap the series before differentiating.

For 2D data (no position_z assay) the default normal is the global z axis, so a positive angle means the distal vector is counter-clockwise from the proximal vector in the x-y plane. For 3D data with plane_normal = NULL the normal is the least-variance direction of the two joint vectors over all frames (a best-fit plane), which is only defined up to a global flip; the function warns in that case because the flexion(+) direction is then a property of the data rather than a clinical convention. Because v_i is projected onto the plane, out-of-plane motion makes the signed magnitude smaller than the unsigned 3D angle; the two agree exactly for planar motion.

For "groodsuntay" / "ISB", a proximal and distal anatomical frame are built for each joint (via jointCoordinateSystem()) and the three signed angles are extracted with groodSuntayAngles().

This is a *sequence- dependent* Grood-Suntay floating-axis (Z-X-Y Cardan) decomposition: flexion (about the proximal medio-lateral axis) is extracted first and internal/ external rotation (about the distal long axis) last.

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

```
demo <- demoMoCapData(seed = 1)
pe_ang <- calculateJointAngles(demo$mocap,
  joints = list(knee_r = list(proximal = "Pelvis_R",
                              joint = "Knee_R", distal = "Ankle_R")))
SummarizedExperiment::assayNames(pe_ang)
```

---

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

```
n <- 50
mk <- function(v) { m <- matrix(v, n, 3); colnames(m) <- c("A", "B", "C"); m }
pe <- PhysioExperiment(
  assays = S4Vectors::SimpleList(position_x = mk(1), position_y = mk(2),
                                 position_z = mk(0)),
  colData = S4Vectors::DataFrame(label = c("A", "B", "C"),
                                 type = rep("marker", 3)),
  samplingRate = 100)
seg_com <- calculateSegmentCOM(pe, proximal_markers = "A",
  distal_markers = "B", com_fractions = 0.5)
names(seg_com)
head(seg_com[[1]])
```

`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)
```

---

classifyGaitPatterns *Classify gait pathology patterns from a bundle of gait variables*

Description

Runs each applicable detector on the supplied gait-cycle waveforms and returns a per-pattern flag and severity. A pattern is evaluated only when its input variable is present in `variables`.

Usage

```
classifyGaitPatterns(variables, params = list())
```

Arguments

variables	A named list of gait-cycle waveforms. Recognised names: knee_flexion, ankle_power, foot_ml, pelvic_obliquity.
params	Optional named list of per-pattern argument lists forwarded to the individual detectors (e.g. list(stiff_knee = list(threshold = 40))).

Value

A data frame (class gait_pattern_classification) with columns pattern, flagged, severity, metric, threshold, one row per evaluated pattern.

See Also

gaitPatternLibrary()

Examples

```
vars <- list(knee_flexion = 60 * sin(seq(0, pi, length.out = 101)))
classifyGaitPatterns(vars)
```

---

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()

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phases <- segmentPhases(data, events, schema_gait)
combineTrials(phases, phases, labels = c("t1", "t2"))
```

---

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

```
demo <- demoMoCapData(seed = 1)
pe_acc <- computeAcceleration(demo$mocap)
SummarizedExperiment::assayNames(pe_acc)
```

---

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

```
demo <- demoMoCapData(seed = 1)
pe_jerk <- computeJerk(demo$mocap)
SummarizedExperiment::assayNames(pe_jerk)
```

computeJointPower *Compute joint power from moment and angular velocity*

Description

Joint power is the product of the net joint moment and the joint angular velocity. Positive power (`generation`) reflects concentric muscle action doing work on the segment; negative power (`absorption`) reflects eccentric action absorbing energy. Set `split = TRUE` to return the generation and absorption components alongside the total.

Usage

```
computeJointPower(moment, angular_velocity, split = FALSE)
```

Arguments

`moment`	Numeric vector of joint moment values.
`angular_velocity`	Numeric vector of joint angular velocity in rad/s.
`split`	Logical; if `TRUE`, return a data frame with the total `power` split into non-negative `generation` (`P > 0`) and non-positive `absorption` (`P < 0`) components. Default `FALSE` returns the total power vector (backward compatible).

Value

If `split = FALSE`, a numeric vector of joint power (W). If `split = TRUE`, a data frame with columns `power`, `generation`, and `absorption` (each in W); `generation + absorption == power` element-wise.

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, `jointWork()` for integrating power into concentric/eccentric work, `labelPowerBursts()` for Winter power-burst nomenclature.

Examples

```
computeJointPower(c(10, 12, 8), c(2, 1.5, 1))
computeJointPower(c(10, -12, 8), c(2, 1.5, -1), split = TRUE)
```

`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

```
demo <- demoMoCapData(seed = 1)
pe_speed <- computeSpeed(computeVelocity(demo$mocap))
SummarizedExperiment::assayNames(pe_speed)
```

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

```
demo <- demoMoCapData(seed = 1)
pe_vel <- computeVelocity(demo$mocap)
SummarizedExperiment::assayNames(pe_vel)
```

`continuousRelativePhase` *Continuous relative phase between two joints*

Description

The phase-plane coordination measure (Hamill et al. 2000): each joint angle is mapped to a phase angle from its (normalised) phase portrait, and the continuous relative phase is their difference through the movement. Its mean absolute value (MARP) summarises whether the joints move more in-phase (near 0) or out-of-phase (near 180).

Usage

```
continuousRelativePhase(angle1, angle2, normalize = TRUE)
```

Arguments

`angle1`, `angle2`	Numeric joint-angle time series of equal length, one movement cycle (typically time-normalised); `angle1` is the reference (e.g. proximal) joint.
`normalize`	Normalise the phase portrait (recommended; corrects for amplitude differences).

Value

a `crp_result` list: `crp` (degrees, wrapped to +/-180), `phase1`, `phase2` (phase angles, degrees) and `marp` (mean absolute relative phase).

References

Hamill J, et al. (2000) Clin Biomech 15:S31-S39.

See Also

`vectorCoding()`, `coordinationVariability()`

Examples

```
t <- seq(0, 2 * pi, length.out = 101)
continuousRelativePhase(sin(t), sin(t))$marp        # in-phase ~ 0
```

`contractionWork` *Angular work and impulse for strength-test repetitions*

Description

Integrates torque over time to obtain angular impulse and, when angle or angular velocity is available, integrates torque over angular displacement to obtain mechanical work.

Usage

```
contractionWork(
  torque,
  sampling_rate,
  angle = NULL,
  angular_velocity = NULL,
  reps = NULL,
  onset_threshold = 7.5,
  body_mass = NULL
)
```

Arguments

`torque`	Numeric torque trace in N m.
`sampling_rate`	Sampling rate in Hz.
`angle`	Optional angle trace in degrees.
`angular_velocity`	Optional angular-velocity trace in degrees/s, used when `angle` is not supplied.
`reps`	Optional named list of strictly increasing integer sample-index vectors. When `NULL`, repetitions are segmented with `computeImpulse()`.
`onset_threshold`	Non-negative torque threshold used for automatic repetition segmentation.
`body_mass`	Optional positive body mass in kg.

Value

A `contraction_work` data frame with angular impulse (N m s), work (J), and optional mass-normalised values for each repetition.

References

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

Examples

```
torque <- c(seq(0, 100, length.out = 51), seq(98, 0, length.out = 50))
contractionWork(torque, sampling_rate = 100,
                reps = list(rep1 = seq_along(torque)))
```

`coordinationVariability`

Inter-joint coordination variability across cycles

Description

The trial-to-trial (cycle-to-cycle) variability of the coordination measure – the deviation phase for continuous relative phase (linear SD) or the coupling- angle circular SD for vector coding – computed at each point of the (time-normalised) cycle. Coordination variability is a motor-control marker: abnormally high or low values accompany injury and neurological disease.

Usage

```
coordinationVariability(
  angle1_cycles,
  angle2_cycles,
  method = c("crp", "vector_coding")
)
```

Arguments

`angle1_cycles`, `angle2_cycles`
: Numeric matrices `cycles x points` (each row one time-normalised cycle) of the two joints' angles.

`method`
: `"crp"` (deviation phase, linear SD) or `"vector_coding"` (coupling-angle circular SD).

Value

a `coordination_variability` list: `method`, `variability` (per-point), `mean_variability` and `measure` (the per-cycle coordination matrix).

References

Hamill (2000); Needham (2014).

See Also

`continuousRelativePhase()`, `vectorCoding()`

Examples

```
m <- matrix(sin(seq(0, 2 * pi, length.out = 50)), nrow = 5, ncol = 50,
            byrow = TRUE)
coordinationVariability(m, m)$mean_variability      # identical cycles -> 0
```

`copFromPressure` *Derive center of pressure from a pressure movie*

Description

Computes the pressure-weighted centroid at each frame. Grid columns define the mediolateral (`cop_x`) direction and rows define the anteroposterior (`cop_y`) direction, matching `calculateCOP()` and `swayMetrics()`.

Usage

```
copFromPressure(pm, contact_threshold = 0)
```

Arguments

`pm`	A `pressure_movie`.
`contact_threshold`	Pressure threshold in kPa. Values at or below the threshold do not contribute to force or center of pressure.

Value

A data frame with `time`, `cop_x`, `cop_y`, `total_force`, and `contact_area`. CoP coordinates are in mm and force is in N.

References

Pataky TC (2012). Spatial resolution in plantar pressure measurement revisited. *Journal of Biomechanics*, 45:2116-2124.

See Also

`swayMetrics()`, `calculateCOP()`

Examples

```
x <- array(0, c(3, 3, 2))
x[2, 3, ] <- 100
copFromPressure(pressureMovie(x, 100, dx = 5, dy = 5))
```

`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

```
demo <- demoMoCapData(seed = 1)
swaps <- detectSwaps(demo$mocap)
pe_fixed <- correctSwaps(demo$mocap, swaps)
pe_fixed
```

`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.

Requires a native OpenSim installation (`SystemRequirements: OpenSim`) and setup/model files, so the example cannot run offline and is not executed.

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)
```

`detectCircumduction` *Detect circumduction (excessive lateral swing trajectory)*

Description

Circumduction is a lateral (outward) swing of the foot/hip used to advance a functionally long limb (e.g. with stiff knee or drop foot). It is quantified as the peak lateral excursion of the foot (or hip) during swing relative to the stance-phase baseline.

Usage

```
detectCircumduction(
  foot_ml,
  swing = c(60, 100),
  stance = c(0, 55),
  threshold = 0.04,
  severe = 0.12,
  lateral_sign = 1
)
```

Arguments

`foot_ml`	Medio-lateral foot (or hip) position over one gait cycle, in metres, with lateral being positive (see `lateral_sign`).
`swing, stance`	Percentage windows of swing and stance (defaults `c(60, 100)` and `c(0, 55)`).
`threshold`	Lateral excursion (m) above which circumduction is flagged (default 0.04).
`severe`	Lateral excursion mapped to severity 1 (default 0.12 m).
`lateral_sign`	+1 if positive `foot_ml` is lateral, -1 otherwise.

Value

A `gait_pattern_flag` object.

References

Perry J, Burnfield JM (2010); Kerrigan DC, et al. (2000).

See Also

classifyGaitPatterns()

Examples

```
pct <- seq(0, 100, length.out = 101)
foot_ml <- rep(0, 101)
swing <- pct >= 60
foot_ml[swing] <- 0.10 * sin(pi * (pct[swing] - 60) / 40)
detectCircumduction(foot_ml)
```

---

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", "zeni"),
  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)
```

---

detectEventsZeni *Detect gait events with the Zeni coordinate method*

Description

Implements the coordinate-based gait-event detector of Zeni, Richards & Higginson (2008). Heel strike (initial contact, IC) occurs when a heel marker reaches its most anterior position relative to a pelvis/COM reference; toe off (TO) occurs when a toe marker reaches its most posterior position relative to the same reference. Because events are keyed to the foot marker *relative to* the pelvis, the method works on treadmill trials where the body does not translate in the lab frame, and it returns per-stride IC/TO for both sides across a full trial.

Usage

```
detectEventsZeni(
  pe,
  markers,
  reference = "sacrum",
  ap_axis = NULL,
  direction = NULL,
  body_mass = NULL,
  min_separation = NULL,
  sampling_rate = NULL
)
```

Arguments

pe	A PhysioExperiment with marker position_x/position_y (and optionally position_z) assays (frames x markers).
markers	A named list of marker names (columns of the position assays) with entries heel_left, heel_right, toe_left, toe_right. A side is skipped if both its heel and toe entries are NULL/missing.
reference	Progression reference: a marker name (e.g. a sacral marker, the default "sacrum"), or "com" to use the whole-body centre of mass from calculateCOM() (requires body_mass).
ap_axis	Anterior-posterior (progression) axis: "x", "y", "z" or 1/2/3. NULL (default) auto-detects the axis with the largest heel-relative-to-reference excursion.
direction	+1 or -1 to orient the AP axis so anterior is positive; NULL (default) auto-detects it (the fastest relative foot motion, i.e. swing, points anterior).
body_mass	Body mass in kg, required only when reference = "com".
min_separation	Minimum number of frames between successive same-type events (refractory period). NULL (default) estimates it from the dominant stride period via autocorrelation.
sampling_rate	Sampling rate in Hz; defaults to samplingRate(pe).

Details

The result is a detected_events data frame compatible with calculateGaitParameters(); heel-strike/toe-off rows are named left_heel_strike/right_heel_strike/left_toe_off/right_toe_off and carry an explicit side column.

Value

A detected_events data frame with one row per detected event (columns event, label, index, time, percent, method, confidence, side), sorted by frame index.

References

Zeni JA, Richards JG, Higginson JS (2008). "Two simple methods for determining gait events during treadmill and overground walking." Gait & Posture 27(4):710-714.

See Also

detectEvents() (use method = "zeni"), calculateGaitParameters(), calculateCOM().

Examples

```
demo <- demoMoCapData(seed = 1)
ev <- detectEventsZeni(demo$mocap,
  markers = list(heel_right = "Ankle_R", toe_right = "Toe_R",
                 heel_left = "Ankle_L", toe_left = "Toe_L"),
  reference = "Pelvis_R")
head(ev)
```

---

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 <- PhysioExperiment::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")
```

---

detectPushOffDeficit *Detect push-off deficit (reduced ankle A2 power / plantarflexor output)*

Description

Reduced ankle plantarflexor push-off is quantified from the peak positive ankle power in late stance (the A2 burst; normal $\approx 3-4$ W/kg) or, equivalently, the peak late-stance plantarflexor moment.

Usage

```
detectPushOffDeficit(
  ankle_power,
  window = c(40, 62),
  normal_peak = 3.5,
  threshold = 2,
  metric_name = "peak_ankle_A2_power"
)
```

Arguments

`ankle_power`	Ankle joint power over one gait cycle. Interpreted as W/kg (or the peak ankle plantarflexor moment if `metric_name` is set).
`window`	Percentage window of late stance / push-off (default `c(40, 62)`).
`normal_peak`	Normal peak A2 power (default 3.5 W/kg).
`threshold`	Peak below which a deficit is flagged (default 2.0).
`metric_name`	Label for the metric (default `"peak_ankle_A2_power"`).

Value

A `gait_pattern_flag` object.

References

Perry J, Burnfield JM (2010); Winter DA (2009).

See Also

`classifyGaitPatterns()`

Examples

```
pct <- seq(0, 100, length.out = 101)
ankle_power <- -0.6 * exp(-((pct - 15) / 8)^2)
push <- pct > 40 & pct < 62
ankle_power[push] <- 1.0 * sin(pi * (pct[push] - 40) / 22)
detectPushOffDeficit(ankle_power)
```

---

detectStanceZUPT *Detect foot-IMU stance phases for zero-velocity updates*

Description

Flags the stationary (foot-flat) samples of a foot-mounted IMU trace, used as zero-velocity-update (ZUPT) anchors by `strapdownIntegrate()` and `footImuGait()`. The default `"magnitude"` detector marks a sample as stance when the accelerometer magnitude is close to gravity *and* the gyroscope magnitude is small; the `"shoe"` detector is the SHOE (Stance Hypothesis Optimal Estimation) generalised-likelihood-ratio test of Skog et al. (2010).

Usage

```
detectStanceZUPT(
  accel,
  gyro,
  sampling_rate,
  g = 9.81,
  method = c("magnitude", "shoe"),
  accel_threshold = 0.6,
  gyro_threshold = 0.6,
  sigma_a = 0.05,
  sigma_g = 0.05,
  gamma = 10000,
  window = NULL,
  min_stance = NULL
)
```

Arguments

`accel`	Numeric matrix (n x 3) of accelerometer readings (m/s^2).
`gyro`	Numeric matrix (n x 3) of gyroscope readings (rad/s).
`sampling_rate`	Sampling rate in Hz.
`g`	Gravitational acceleration magnitude (default 9.81 m/s^2).
`method`	`"magnitude"` (default) or `"shoe"`.
`accel_threshold`	Magnitude method: max $\lvert \lVert$`accel`$\rVert$ `- g`$\rvert$ for stance (default 0.6 m/s^2).
`gyro_threshold`	Magnitude method: max $\lVert$`gyro`$\rVert$ for stance (default 0.6 rad/s).

`sigma_a`, `sigma_g`
: SHOE method: accelerometer / gyroscope noise standard deviations (default 0.05 and 0.05).

`gamma`
: SHOE method: test-statistic threshold below which a sample is stance (default 1e4).

`window`
: Number of samples in the SHOE sliding window (default is about 50 ms of data).

`min_stance`
: Minimum stance-run length in samples; shorter runs are discarded as spurious (default is about 50 ms of data).

Value

A logical vector of length `n`, `TRUE` at stance samples.

References

Skog I, Handel P, Nilsson J-O, Rantakokko J (2010). "Zero-velocity detection – an algorithm evaluation." IEEE Trans Biomed Eng 57(11):2657-2666.

See Also

`strapdownIntegrate()`, `footImuGait()`.

Examples

```
set.seed(1)
g <- 9.81
stance_block <- matrix(rep(c(0, 0, -g), each = 60), 60, 3)
swing_block <- matrix(rnorm(40 * 3, 0, 3), 40, 3)
accel <- rbind(stance_block, swing_block, stance_block, swing_block, stance_block)
gyro <- rbind(matrix(0, 60, 3), matrix(rnorm(40 * 3, 0, 1.5), 40, 3),
              matrix(0, 60, 3), matrix(rnorm(40 * 3, 0, 1.5), 40, 3),
              matrix(0, 60, 3))
det <- detectStanceZUPT(accel, gyro, sampling_rate = 200)
sum(det)
```

`detectStiffKnee` *Detect stiff-knee gait (reduced / delayed peak swing knee flexion)*

Description

Stiff-knee gait is marked by reduced peak knee flexion in swing (normal $\approx 60^\circ$ near 73\ (Perry & Burnfield; Goldberg et al. 2006).

Usage

```
detectStiffKnee(
  knee_flexion,
  swing = c(60, 100),
  normal_peak = 60,
  threshold = 45,
  severe_floor = 10,
  delay_threshold = 80
)
```

Arguments

knee_flexion	Knee flexion angle (degrees, flexion positive) over one normalised gait cycle.
swing	Percentage window of swing phase (default c(60, 100)).
normal_peak	Normal peak swing knee flexion in degrees (default 60).
threshold	Peak below which stiff-knee is flagged (default 45).
severe_floor	Peak flexion mapped to severity 1 (default 10).
delay_threshold	Peak-timing percentage above which the peak is considered delayed (default 80).

Value

A gait_pattern_flag object.

References

Perry J, Burnfield JM (2010); Goldberg SR, et al. (2006).

See Also

classifyGaitPatterns(), gaitPatternLibrary()

Examples

```
# normal knee flexion peaks ~60 deg in swing
k <- c(seq(5, 18, length.out = 30), seq(18, 5, length.out = 30),
       60 * sin(seq(0, pi, length.out = 41)))
detectStiffKnee(k)
```

`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

```
demo <- demoMoCapData(seed = 1)
swaps <- detectSwaps(demo$mocap)
swaps
```

`detectTrendelenburg` *Detect Trendelenburg sign (excessive contralateral pelvic drop)*

Description

A Trendelenburg gait reflects hip-abductor weakness on the stance limb: the contralateral (swing-side) pelvis drops during single-limb stance beyond the normal small obliquity ($\approx 4-5^\circ$).

Usage

```
detectTrendelenburg(
  pelvic_obliquity,
  stance = c(10, 50),
  threshold = 5,
  severe = 15,
  drop_sign = 1
)
```

Arguments

`pelvic_obliquity`	Pelvic obliquity (degrees) over one gait cycle, with the contralateral drop being positive (see `drop_sign`).
`stance`	Percentage window of stance (default `c(10, 50)`, single-limb).
`threshold`	Drop (deg) above which Trendelenburg is flagged (default 5).
`severe`	Drop mapped to severity 1 (default 15 deg).
`drop_sign`	`+1` if positive obliquity is a contralateral drop, `-1` otherwise.

Value

A `gait_pattern_flag` object.

References

Perry J, Burnfield JM (2010).

See Also

`classifyGaitPatterns()`

Examples

```
pct <- seq(0, 100, length.out = 101)
pelvic_obliquity <- rep(0, 101)
pelvic_obliquity[pct >= 10 & pct <= 50] <- 12
detectTrendelenburg(pelvic_obliquity)
```

`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)
```

---

dimensionlessJerk *Dimensionless jerk of a speed profile*

Description

Dimensionless jerk of a speed profile

Usage

```
dimensionlessJerk(speed, fs)
```

Arguments

speed	Numeric speed profile.
fs	Sampling frequency in Hz.

Value

The (positive) dimensionless jerk. See ldlj for the log form.

Examples

```
set.seed(1)
speed <- abs(sin(seq(0, pi, length.out = 200))) + 0.01
dimensionlessJerk(speed, fs = 120)
```

---

dmpFit *Fit a Dynamic Movement Primitive to a demonstration*

Description

Learns the forcing term of a DMP from one demonstrated trajectory, so it can later be regenerated (and generalised to a new goal or duration) by dmpGenerate().

Usage

```
dmpFit(y, times = NULL, n_basis = 25L, alpha_z = 25)
```

Arguments

y	Demonstration: a numeric vector (1-D) or T x D matrix (per-column DOF).
times	Optional time stamps (length T); defaults to 0..1.
n_basis	Number of Gaussian basis functions (default 25).
alpha_z	Attractor gain (default 25; damping beta_z = alpha_z / 4, critically damped).

Value

a dmp object: learned weights (n_basis x D), goal, start, duration, and parameters.

References

Ijspeert AJ, et al. (2013) Neural Comput 25:328-373.

See Also

dmpGenerate(), promFit()

Examples

```
t <- seq(0, 1, length.out = 100)
y <- 10 * (10*t^3 - 15*t^4 + 6*t^5)              # min-jerk demo
d <- dmpFit(y); rg <- dmpGenerate(d)
max(abs(rg$y[, 1] - y))                           # reproduces the demo
```

---

dmpGenerate *Generate a trajectory from a Dynamic Movement Primitive*

Description

Integrates a fitted dmpFit() DMP, optionally generalising to a new goal, start or duration (temporal scaling) while preserving the movement's shape and guaranteeing convergence to the goal.

Usage

```
dmpGenerate(dmp, goal = NULL, start = NULL, tau = NULL, n = 100L)
```

Arguments

dmp	A dmp from dmpFit().
goal, start	Optional new goal / start (length D); default the learned ones.
tau	Optional new duration (temporal scaling); default the learned duration.
n	Number of output samples (default 100).

Value

a dmp_trajectory: time, y, yd (n x D).

See Also

dmpFit()

Examples

```
t <- seq(0, 1, length.out = 100)
y <- 10 * (10 * t^3 - 15 * t^4 + 6 * t^5)   # minimum-jerk demo
d <- dmpFit(y)
rollout <- dmpGenerate(d)
dim(rollout$y)
```

---

downloadOpenCapModel *Download the OpenSim Model for an OpenCap Session*

Description

Downloads the subject-specific, scaled OpenSim model (`.osim`) that OpenCap produced for a session. This model is the input the local OpenSim toolchain needs (runOpenSimFromMarkers / runOpenSimFromOpenCap): OpenCap builds and scales it in the cloud from the neutral/calibration trial, and **PhysioOpenSim** provides no scaling tool of its own, so the model must come from OpenCap.

Usage

```
downloadOpenCapModel(
  session_id,
  trial_id = NULL,
  api_key = NULL,
  base_url = "https://app.opencap.ai/api",
  dest = NULL
)
```

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.
base_url	Character string giving the base URL for the OpenCap API. Default is "https://app.opencap.ai/api".
dest	Character path to write the model to. If NULL (default) a temporary .osim file is created.

Details

Requires the **httr** package and an OpenCap API key (see `readOpenCap`). The model URL is taken from the session metadata when present and otherwise constructed from the documented session endpoint.

Requires the OpenCap web API (network access and an API key), so the example cannot run offline and is not executed.

Value

The local path to the downloaded `.osim` model (invisibly).

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

`readOpenCap()`, `runOpenSimFromOpenCap()`, `runOpenSimFromMarkers()`

Examples

```
## Not run:
osim <- downloadOpenCapModel("abcd1234-5678-90ab-cdef-1234567890ab")

## End(Not run)
```

`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()

Examples

```
x <- sin(seq(0, pi, length.out = 100)) * 30
y <- sin(seq(0, pi, length.out = 100) + 0.3) * 30
dr <- dtwDistance(x, y)
warped <- dtwWarp(x, dr)
length(warped)
```

endpointError *Reaching endpoint accuracy and precision*

Description

Computes constant, variable, absolute, and root-mean-square endpoint error. Variable error uses the population RMS distance about the endpoint centroid, preserving the identity `RMSE^2 = constant_error^2 + variable_error^2`.

Usage

```
endpointError(endpoints, target)
```

Arguments

`endpoints`	Numeric endpoint vector for a one-dimensional task, or an `n` by 2/3 numeric matrix with one endpoint per row.
`target`	Numeric target scalar or vector matching the endpoint dimension.

Value

A `reaching_endpoint_error` object with constant, variable, absolute, RMS, and Fitts effective-width errors.

References

Hancock GR, Butler MS, Fischman MG (1995). On the problem of two-dimensional error scores: measures and analyses of accuracy, bias, and consistency. *Journal of Motor Behavior*, 27:241-250.

Fitts PM (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381-391. doi:10.1037/h0055392

Examples

```
endpointError(c(11, 12, 13, 12), target = 10)
```

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)
)
```

---

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()

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phases <- segmentPhases(data, events, schema_gait)
extractPhase(phases, getPhaseNames(schema_gait)[1])
```

---

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 <- PhysioExperiment::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

```
if (requireNamespace("signal", quietly = TRUE)) {
  demo <- demoMoCapData(seed = 1)
  pe_filt <- filterSignals(demo$mocap, type = "lowpass", cutoff = 6)
  SummarizedExperiment::assayNames(pe_filt)
}
```

`floquetMultipliers` *Floquet multipliers (orbital stability of a cyclic movement)*

Description

Quantifies how quickly perturbations to a limit-cycle movement (e.g. gait) decay from one stride to the next. At each phase of the cycle a linear Poincare (return) map is fitted between the SAME phase in consecutive strides (`dS_{n+1} = J dS_n`); the Floquet multipliers are its eigenvalues. The maximum modulus < 1 means the movement is orbitally stable (perturbations shrink stride to stride).

Usage

```
floquetMultipliers(states)
```

Arguments

`states`	Either a `[stride, phase, state]` array, or a list of `phase x state` matrices (one per stride). Strides must be consecutive and sampled at the same phases; the state is a vector (e.g. joint angles and velocities).

Value

a `floquet_result`: `max_multiplier` (largest modulus over phases), `multipliers` (eigenvalues of the section return map), `per_phase_max` (the Dingwell per-phase maximum multiplier) and `orbitally_stable`.

References

Hurmuzlu Y, Basdogan C (1994) J Biomech Eng 116:30-36; Dingwell JB, Kang HG (2007) J Biomech Eng 129:586-593.

See Also

`recurrenceQuantification()`, `maxLyapunovExponent()`

Examples

```
set.seed(1)
A <- matrix(c(0.5, 0.1, 0, 0.3), 2, 2)             # stride-to-stride return map
P <- 4; nS <- 80; strides <- array(0, c(nS, P, 2)); d <- rnorm(2)
cyc <- cbind(sin(1:P), cos(1:P))
for (n in 1:nS) { for (k in 1:P) strides[n, k, ] <- cyc[k, ] + d
  d <- as.numeric(A %*% d) + rnorm(2, 0, 0.01) }
floquetMultipliers(strides)$max_multiplier          # ~ 0.5 (< 1, stable)
```

`footImuGait` *Foot-IMU spatiotemporal gait parameters*

Description

End-to-end foot-mounted IMU gait pipeline: estimate orientation, remove gravity, rotate acceleration into the world frame, detect stance, and ZUPT-integrate to a drift-bounded foot trajectory, then derive per-stride stride length, foot clearance, stance/swing time and gait velocity.

Usage

```
footImuGait(
  accel,
  gyro,
  sampling_rate,
  g = 9.81,
  orientation = NULL,
  orientation_method = c("madgwick", "complementary"),
  beta = 0.02,
  stance = NULL,
  vertical_axis = 3,
  ...
)
```

Arguments

`accel`	Numeric matrix (n x 3) of accelerometer readings (m/s^2).
`gyro`	Numeric matrix (n x 3) of gyroscope readings (rad/s).
`sampling_rate`	Sampling rate in Hz.
`g`	Gravitational acceleration magnitude (default 9.81 m/s^2).
`orientation`	Optional orientation (a data frame with `q_w`/`q_x`/`q_y`/ `q_z`, or an n x 4 quaternion matrix). If `NULL`, `estimateOrientation()` is run internally.
`orientation_method`	Sensor-fusion method for the internal orientation estimate, `"madgwick"` (default) or `"complementary"`. Ignored when `orientation` is supplied.
`beta`	Madgwick filter gain for the internal orientation estimate (default 0.02). A foot IMU sees large dynamic accelerations, so a small, gyro-dominant gain avoids swing-phase tilt error; raise it for lower-dynamic mounting or noisier gyroscopes. Ignored when `orientation` is supplied.
`stance`	Optional logical stance vector; if `NULL`, `detectStanceZUPT()` is used.
`vertical_axis`	World axis that points up (1, 2 or 3; default 3). Used for foot clearance and to de-drift vertical position to the floor.
`...`	Additional arguments forwarded to `detectStanceZUPT()` (e.g. `method`, `accel_threshold`).

Value

An `imu_gait` object: a list with `strides` (a data frame with one row per detected stride: `stride`, `stride_length`, `foot_clearance`, `stance_time`, `swing_time`, `stride_time`, `gait_velocity`), the world-frame `position` and `velocity` matrices, the `stance` mask, and `sampling_rate`.

References

Mariani B et al. (2010) J Biomech 43(15):2999-3006; Rebula JR, Ojeda LV, Adamczyk PG, Kuo AD (2013). "Measurement of foot placement and its variability with inertial sensors." Gait & Posture 38(4):974-980.

See Also

`detectStanceZUPT()`, `strapdownIntegrate()`, `estimateOrientation()`, `removeGravity()`.

Examples

```
set.seed(1)
g <- 9.81
stance_block <- matrix(rep(c(0, 0, -g), each = 60), 60, 3)
swing_block <- matrix(rnorm(40 * 3, 0, 3), 40, 3)
accel <- rbind(stance_block, swing_block, stance_block, swing_block, stance_block)
gyro <- rbind(matrix(0, 60, 3), matrix(rnorm(40 * 3, 0, 1.5), 40, 3),
              matrix(0, 60, 3), matrix(rnorm(40 * 3, 0, 1.5), 40, 3),
              matrix(0, 60, 3))
res <- footImuGait(accel, gyro, sampling_rate = 200)
names(res)
```

---

forwardDynamics *Forward dynamics: integrate the equations of motion*

Description

Given a limb model, initial state and joint torques, integrates the motion with a fixed-step 4th-order Runge-Kutta scheme.

Usage

```
forwardDynamics(model, q0, qd0, torque = NULL, dt = 0.001, n_steps = 1000L)
```

Arguments

`model`	A `planar_limb_model` (or any list with a `forward_accel(q, qd, tau)` and `ndof`).
`q0`, `qd0`	Initial joint angles and velocities (length `ndof`).
`torque`	Joint torques: `NULL` (free/passive), a length-`ndof` constant, an `n_steps x ndof` matrix, or a function `(t, q, qd) -> torque`.
`dt`	Time step (s).
`n_steps`	Number of integration steps.

Value

an `fd_result`: `time`, `q`, `qd` (`(n_steps+1) x ndof`), `tau`, and `energy` (total mechanical energy per step).

See Also

`planarLimbModel()`, `predictiveSimulation()`

Examples

```
m <- planarLimbModel(c(7, 3), c(0.4, 0.4))
fd <- forwardDynamics(m, q0 = c(pi/2 - 0.3, 0.2), qd0 = c(0, 0),
                      torque = NULL, dt = 0.001, n_steps = 500)
diff(range(fd$energy)) / mean(abs(fd$energy))     # ~ 0: energy conserved
```

---

forwardKinematics2D *Forward kinematics of a planar chain*

Description

Node positions of a serial planar chain from a base, the fixed segment lengths and the ABSOLUTE segment angles (radians, from the x-axis).

Usage

```
forwardKinematics2D(angles, lengths, base = c(0, 0))
```

Arguments

`angles`	Absolute segment angles (length `n_segments`).
`lengths`	Segment lengths (length `n_segments`).
`base`	Root position (length 2; default origin).

Value

a `(n_segments + 1) x 2` matrix of node positions.

See Also

`inverseKinematicsMarkers()`

Examples

```
forwardKinematics2D(c(0, pi/2), c(1, 1))          # right angle
```

gaitAtaxiaIndex *Gait-ataxia composite from gait-variability sub-metrics*

Description

Aggregates the standard gait-ataxia sub-metrics (each higher = more ataxic): the coefficients of variation of step width, stride length and stride time (from `summarizeGaitParameters()`, the `cv` column, in per cent) and trunk sway (e.g. `cop_path_length` from `swayMetrics()`, or trunk-marker RMS).

Usage

```
gaitAtaxiaIndex(
  step_width_cv = NA_real_,
  stride_length_cv = NA_real_,
  stride_time_cv = NA_real_,
  trunk_sway = NA_real_,
  reference = NULL,
  weights = NULL
)
```

Arguments

`step_width_cv`, `stride_length_cv`, `stride_time_cv`
: Gait coefficients of variation (per cent); any may be `NA`.

`trunk_sway`
: Trunk-sway magnitude (higher = worse); optional.

`reference`, `weights`
: As in `limbAtaxiaIndex()`.

Value

An S3 `gait_ataxia` list: `metrics`, per-metric `z`, and `composite`.

See Also

`summarizeGaitParameters()`, `swayMetrics()`, `limbAtaxiaIndex()`

Examples

```
gaitAtaxiaIndex(step_width_cv = 28, stride_length_cv = 9, stride_time_cv = 6)
```

`gaitDeviationIndex` *Gait Deviation Index (Schwartz & Rozumalski 2008)*

Description

Scores overall gait pathology on a scale where the normative control population averages 100 with a standard deviation of 10 (by construction); values below 100 indicate greater deviation, each 10 points corresponding to one standard deviation. The subject's nine kinematic curves are projected onto the normative feature basis, and the log of the distance to the control mean is standardised against the control distribution.

Usage

```
gaitDeviationIndex(kinematics, norm = NULL, basis = NULL, n_features = 15L)
```

Arguments

`kinematics`	Subject kinematics (see `gaitVariableScore()`); curves are time-normalised to the basis's point count (typically 51).
`norm`	A `gait_norm` reference (from `PhysioGaitNorm::loadGaitNorm()`); if `NULL`, loaded from PhysioGaitNorm. Ignored when `basis` is supplied.
`basis`	An optional precomputed `gdiBasis()` (reuse across many subjects); if `NULL` it is built from `norm`.
`n_features`	Number of gait features when building the basis (default 15).

Value

A `gait_deviation_index` object: a list with `gdi` (numeric), `distance`, `z`, and `n_features`.

References

Schwartz MH, Rozumalski A (2008). Gait & Posture 28(3):351-357.

See Also

`gdiBasis()`, `gaitProfileScore()`

Examples

```
if (requireNamespace("PhysioGaitNorm", quietly = TRUE)) {
  basis <- gdiBasis()
  vars <- c("pelvic_tilt", "pelvic_obliquity", "pelvic_rotation",
            "hip_flexion", "hip_adduction", "hip_rotation",
            "knee_flexion", "ankle_dorsiflexion", "foot_progression")
  set.seed(1)
  kin <- matrix(rnorm(9 * basis$n_points, 0, 5), 9, basis$n_points,
                dimnames = list(vars, NULL))
  gaitDeviationIndex(kin, basis = basis)$gdi
}
```

gaitPatternLibrary *Registry of supported gait pathology patterns*

Description

Documents the gait deviation patterns detected by this package, their input variable, detector, threshold and clinical reference.

Usage

```
gaitPatternLibrary()
```

Value

A data frame (class `gait_pattern_library`).

See Also

`classifyGaitPatterns()`, `detectStiffKnee()`, `detectPushOffDeficit()`, `detectCircumduction()`, `detectTrendelenburg()`

Examples

```
gaitPatternLibrary()
```

gaitProfileScore *Gait Profile Score (overall RMS deviation from the norm)*

Description

The Gait Profile Score (Baker et al. 2009) summarises overall gait deviation as the root-mean-square of the Gait Variable Scores across all kinematic variables (degrees). It is always non-negative and is zero when the subject equals the normative mean.

Usage

```
gaitProfileScore(kinematics, norm = NULL)
```

Arguments

`kinematics`	Subject kinematics: a numeric matrix with one row per kinematic variable (row names matching `norm$variables`) and one column per gait-cycle point, or a named list of per-variable curves. Curves are time-normalised to the norm's cycle length via `normalizeMovement()`.
`norm`	A `gait_norm` normative reference (from `PhysioGaitNorm::loadGaitNorm()`); if `NULL`, the default reference is loaded from PhysioGaitNorm.

Value

A single non-negative numeric (degrees).

References

Baker R, et al. (2009). Gait & Posture 30(3):265-269.

See Also

`gaitVariableScore()`, `movementAnalysisProfile()`

Examples

```
norm <- list(variables = c("a", "b"),
             mean = matrix(0, 2, 51, dimnames = list(c("a", "b"), NULL)),
             cycle_length = 51)
gaitProfileScore(matrix(1, 2, 51, dimnames = list(c("a", "b"), NULL)), norm)
```

---

gaitVariableScore *Gait Variable Score (per-variable RMS deviation from the norm)*

Description

The Gait Variable Score (Baker et al. 2009) is the root-mean-square difference, over the gait cycle, between a subject's kinematic curve and the normative mean, computed separately for each kinematic variable (degrees).

Usage

```
gaitVariableScore(kinematics, norm = NULL)
```

Arguments

`kinematics`	Subject kinematics: a numeric matrix with one row per kinematic variable (row names matching `norm$variables`) and one column per gait-cycle point, or a named list of per-variable curves. Curves are time-normalised to the norm's cycle length via `normalizeMovement()`.
`norm`	A `gait_norm` normative reference (from `PhysioGaitNorm::loadGaitNorm()`); if `NULL`, the default reference is loaded from PhysioGaitNorm.

Value

A named numeric vector of GVS values (degrees), one per variable.

References

Baker R, et al. (2009). Gait & Posture 30(3):265-269.

See Also

gaitProfileScore(), movementAnalysisProfile()

Examples

```
norm <- list(variables = c("a", "b"),
             mean = matrix(0, 2, 51, dimnames = list(c("a", "b"), NULL)),
             cycle_length = 51)
gaitVariableScore(matrix(1, 2, 51, dimnames = list(c("a", "b"), NULL)), norm)
```

---

gdiBasis *Build a Gait Deviation Index feature basis from a normative reference*

Description

Computes the singular-value-decomposition feature basis of the normative feature population used to score the Gait Deviation Index (Schwartz & Rozumalski 2008), together with the control-population log-distance mean and standard deviation used to standardise the index.

Usage

```
gdiBasis(norm = NULL, n_features = 15L)
```

Arguments

norm	A gait_norm reference with a features matrix (subjects x concatenated kinematics); if NULL, loaded from PhysioGaitNorm.
n_features	Number of singular vectors (gait features) to retain (default 15).

Value

A gdi_basis object.

References

Schwartz MH, Rozumalski A (2008). Gait & Posture 28(3):351-357.

See Also

gaitDeviationIndex()

Examples

```
if (requireNamespace("PhysioGaitNorm", quietly = TRUE)) {
  basis <- gdiBasis()
  basis$n_features
}
```

`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

```
sk <- define_skeleton("BODY_25")
labels <- sk$keypoints$label
set.seed(42)
nk <- length(labels); nf <- 10
mk <- function(b) matrix(rep(b, each = nf), nf)
px <- mk(rnorm(nk, 0, 0.3)); py <- mk(rnorm(nk, 0, 0.3))
pz <- mk(seq(1.8, 0, length.out = nk))
colnames(px) <- colnames(py) <- colnames(pz) <- labels
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    position_x = px, position_y = py, position_z = pz),
  colData = S4Vectors::DataFrame(label = labels, type = rep("keypoint", nk)),
  samplingRate = 30)
get_segment_lengths(pe, sk)
```

---

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()`

Examples

```
getEvent(schema_gait, getEventNames(schema_gait)[1])
```

---

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()`

Examples

```
getEventNames(schema_gait)
```

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()

Examples

```
getPhase(schema_gait, getPhaseNames(schema_gait)[1])
```

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()

Examples

```
getPhaseColors(schema_gait)
```

---

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()

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phases <- segmentPhases(data, events, schema_gait)
head(getPhaseData(phases))
```

---

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()

Examples

```
getPhaseNames(schema_gait)
```

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")
```

goalEquivalentManifold *Goal-Equivalent Manifold (GEM) decomposition for a scalar goal*

Description

The scalar-goal special case of UCM (Cusumano & Cesari 2006): given the gradient of a scalar goal function, split the execution variability into goal-equivalent (along the level set – does not change the goal) and non-goal-equivalent (along the gradient – changes it), and report the motor-equivalent ratio.

Usage

```
goalEquivalentManifold(execution, goal_gradient = NULL, goal = NULL)
```

Arguments

execution	An N x m matrix of execution/body variables across trials.
goal_gradient	Length-m gradient of the goal function at the mean (the direction in which the goal changes fastest). Provide this, or goal.
goal	Optional scalar goal function R^m -> R; its gradient is computed numerically at the mean when goal_gradient is not given.

Value

a gem_result list: gev (goal-equivalent variance, along the GEM), ngev (non-goal-equivalent variance, along the gradient), me_ratio (gev / ngev, > 1 = variability channelled into the goal-irrelevant direction), log_me_ratio.

References

Cusumano JP, Cesari P (2006) Biol Cybern 94:367-379.

See Also

uncontrolledManifold()

Examples

```
set.seed(2)
x1 <- rnorm(200, 10, 3); x2 <- 20 - x1 + rnorm(200, 0, 0.4)  # x1+x2 ~ const
goalEquivalentManifold(cbind(x1, x2), goal_gradient = c(1, 1))$me_ratio
```

---

grfLandmarks *Extract stance-phase landmarks from a time-normalised vertical GRF curve*

Description

From a vertical ground-reaction-force curve time-normalised to the stance phase (0-100 %), reads the three classic landmarks of the double-humped stance profile: the loading (weight-acceptance) peak, the mid-stance trough, and the push-off peak. Applied to a body-weight-normalised curve, the landmarks are directly comparable across subjects of different mass, and the peak-trough-peak pattern is the mechanical signature clinical gait analyses contrast between cohorts (e.g. reduced weight-acceptance modulation in pathological gait: lower peaks, a raised trough).

Usage

```
grfLandmarks(
  grf,
  loading_window = 5:40,
  midstance_window = 30:60,
  pushoff_window = 45:90
)
```

Arguments

`grf`	Numeric vector: a vertical GRF curve time-normalised to the stance phase (e.g. 101 points spanning 0-100 %). Usually in body-weight units so the landmarks are subject-mass independent. A matrix (curves x time) is also accepted, in which case the landmarks are read from the ensemble-mean curve (`colMeans`) — the group-mean landmarks of a cohort of stance curves.
`loading_window`, `midstance_window`, `pushoff_window`	Integer index ranges (into `grf`) in which to read the loading peak (a maximum), the mid-stance trough (a minimum) and the push-off peak (a maximum). Defaults target a 101-point stance curve; widen them for populations with atypical timing. The windows deliberately overlap, so the landmarks are found in temporal order: the trough is sought strictly after the loading peak and the push-off peak strictly after the trough, guaranteeing `peak1 < trough < peak2`. A landmark that lands on the upper edge of its window raises a warning, since the true extremum may lie beyond it.

Value

A named numeric vector `c(peak1, trough, peak2)` — the loading peak, the mid-stance trough and the push-off peak, in the units of `grf`.

References

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

Horsak B, Slijepcevic D, Raberger AM, Schwab C, Worisch M, Zeppelzauer M (2020). "GaitRec, a large-scale ground reaction force dataset of healthy and impaired gait." Scientific Data 7:143.

See Also

`filterGRF()` to band-limit the force trace, `computeImpulse()` and `computeLoadingRate()` for other GRF-derived measures.

Examples

```
# a synthetic double-humped stance curve (101 pts, body-weight units)
g <- 1.1 * sin(seq(0, pi, length.out = 101))^0.5
g[46:56] <- g[46:56] * 0.68
grfLandmarks(g)
```

`groodSuntayAngles` *Signed Grood-Suntay / ISB joint angles from segment frames*

Description

Computes the three signed clinical joint angles - flexion/extension, ab/adduction and internal/external rotation - from a proximal and a distal anatomical segment frame, following the Grood-Suntay joint coordinate system (the basis of the ISB recommendations). The flexion axis is the proximal medio-lateral (Z) axis, the rotation axis is the distal long (Y) axis, and the ab/adduction axis is the floating axis perpendicular to both. This is equivalent to a Z-X-Y Cardan decomposition of the relative rotation $R = R_p^\top R_d$; because it is sequence-dependent, the flexion angle is extracted first and the rotation last.

Usage

```
groodSuntayAngles(proximal, distal, degrees = TRUE)
```

Arguments

`proximal`, `distal`	n x 3 x 3 arrays of per-frame segment axes (columns X = antero-posterior, Y = long, Z = medio-lateral), as returned by `jointCoordinateSystem`.
`degrees`	Logical; return degrees (default) or radians.

Value

A numeric n x 3 matrix with columns `flexion`, `abduction` and `rotation`.

References

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

See Also

`jointCoordinateSystem()`, `calculateJointAngles()`

Examples

```
set.seed(1)
proximal <- array(rnorm(100 * 3 * 3), c(100, 3, 3))
distal <- array(rnorm(100 * 3 * 3), c(100, 3, 3))
ang <- groodSuntayAngles(proximal, distal)
head(ang)
```

`hamstringQuadRatio` *Conventional and functional hamstring-to-quadriceps ratios*

Description

Computes the conventional concentric H:Q ratio and a functional ratio that compares eccentric antagonist torque with concentric agonist torque. Optional EMG traces are processed with `processEMG()` and summarised as the hamstring-to-quadriceps mean-envelope ratio.

Usage

```
hamstringQuadRatio(
  quad_con,
  ham_con,
  ham_ecc = NULL,
  quad_ecc = NULL,
  movement = c("extension", "flexion"),
  emg_ham = NULL,
  emg_quad = NULL,
  emg_sampling_rate = NULL
)
```

Arguments

`quad_con`	Concentric quadriceps peak torque or torque trace.
`ham_con`	Concentric hamstring peak torque or torque trace.
`ham_ecc`	Optional eccentric hamstring peak torque or torque trace.
`quad_ecc`	Optional eccentric quadriceps peak torque or torque trace.
`movement`	Movement direction, `"extension"` or `"flexion"`.
`emg_ham, emg_quad`	Optional hamstring and quadriceps EMG traces. Supply both or neither.
`emg_sampling_rate`	EMG sampling rate in Hz, required with EMG traces.

Value

An `hq_ratio` object containing conventional and functional ratios, peak torques, movement direction, and optional EMG ratio.

References

Coombs R, Garbutt G (2002). Developments in the use of the hamstring/ quadriceps ratio for the assessment of muscle balance. *Journal of Sports Science and Medicine*, 1:56-62.

Aagaard P, Simonsen EB, Magnusson SP, Larsson B, Dyhre-Poulsen P (1998). A new concept for isokinetic hamstring:quadriceps muscle strength ratio. *American Journal of Sports Medicine*, 26:231-237. doi:10.1177/03635465980260021201

Examples

```
hamstringQuadRatio(quad_con = 200, ham_con = 120, ham_ecc = 150)
```

`harmonicRatio` *Harmonic ratio of a trunk-acceleration signal*

Description

The harmonic ratio quantifies the smoothness/rhythmicity of trunk acceleration over a stride from its Fourier series: because there are two steps per stride, the even harmonics carry the step-to-step-symmetric (in-phase) content and the odd harmonics the asymmetric content. For anterior-posterior and vertical acceleration the ratio is even/odd (higher = smoother); for medio-lateral it is odd/even (Menz et al. 2003; Bellanca et al. 2013).

Usage

```
harmonicRatio(
  signal,
  n_strides = 1L,
  direction = c("AP", "vertical", "ML"),
  n_harmonics = 20L
)
```

Arguments

`signal`	Numeric acceleration over an integer number of strides (a single stride if `n_strides` = 1).
`n_strides`	Number of strides spanned by `signal` (default 1); the Fourier fundamental is the stride frequency.
`direction`	`"AP"`/`"vertical"` (even/odd) or `"ML"` (odd/even).
`n_harmonics`	Number of harmonics summed (default 20).

Value

A `harmonic_ratio` object with `ratio`, `even_sum`, `odd_sum` and `direction`.

References

Menz HB, et al. (2003); Bellanca JL, et al. (2013).

See Also

`maxLyapunovExponent()`

Examples

```
t <- seq(0, 1, length.out = 200)[-200]
# a clean 2-steps-per-stride (even-harmonic) signal
harmonicRatio(cos(2 * 2 * pi * t), direction = "AP")
```

`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()`

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
hasValidPhases(segmentPhases(data, events, schema_gait))
```

`instrumented10mWT` *Instrumented 10-metre walk test (10mWT)*

Description

Extracts steady-state gait speed (and cadence) over the middle section of a 10-metre walk, excluding the acceleration and deceleration zones, from the forward walking distance over time (e.g. the ZUPT-integrated foot trajectory from `footImuGait()` or a marker distance).

Usage

```
instrumented10mWT(
  position,
  sampling_rate,
  total_distance = 10,
  mid_start = 2,
  mid_end = 8,
  events = NULL,
  events_unit = c("auto", "index", "seconds")
)
```

Arguments

`position`	Cumulative forward distance walked (m) over time.
`sampling_rate`	Sampling rate in Hz.
`total_distance`	Total walkway length in m (default 10).
`mid_start`, `mid_end`	Start/end of the timed steady-state section in m (defaults 2 and 8).
`events`	Optional heel-strike sample indices or times (s) for cadence.
`events_unit`	Unit of `events`: `"auto"` (default) guesses indices vs seconds, or force `"index"` / `"seconds"`. Pass an explicit unit when heel-strike times are whole-number seconds (which `"auto"` would otherwise read as sample indices).

Value

A `walk_test_report` object with `gait_speed` (m/s), `mid_distance`, `mid_time`, `cadence_spm` (or `NA`), and the section boundary times.

References

Standard 10-metre walk test protocol.

See Also

`instrumentedTUG()`, `footImuGait()`

Examples

```
fs <- 100
n <- 12 * fs
position <- seq(0, 10, length.out = n) + rnorm(n, 0, 0.01)
wt <- instrumented10mWT(position, fs)
wt$gait_speed
```

`instrumentedTUG` *Instrumented Timed Up-and-Go (iTUG)*

Description

Segments a Timed Up-and-Go trial from a trunk/lumbar inertial sensor into its sub-phases (sit-to-stand, walk, turn, turn-to-sit) using the angular velocity: turns appear as large excursions of the vertical-axis (yaw) angular velocity, and the postural sit-to-stand / turn-to-sit transitions as the trunk flexion-axis (pitch) events that bracket the walking (Salarian et al. 2010).

Usage

```
instrumentedTUG(
  angular_velocity,
  sampling_rate,
  turn_axis = 3L,
  transition_axis = 2L,
  turn_threshold = NULL,
  min_turn_sec = 0.3
)
```

Arguments

`angular_velocity`	An n x 3 matrix of trunk angular velocity (rad/s).
`sampling_rate`	Sampling rate in Hz.
`turn_axis`	Column index of the vertical/yaw axis (default 3).
`transition_axis`	Column index of the trunk flexion/pitch axis (default 2).
`turn_threshold`	Angular-velocity threshold (rad/s) that marks a turn; `NULL` uses `0.15 * max(abs(yaw))`.
`min_turn_sec`	Minimum turn duration in seconds (default 0.3).

Value

An `itug_report` object with `total_duration`, a `turns` data frame (`start`, `end`, `duration`, `peak_velocity`, `angle_deg`), the `stand_up_time`/`sit_down_time`, and a `phases` data frame.

References

Salarian A, et al. (2010). IEEE Trans Neural Syst Rehabil Eng 18(3):303-310.

See Also

instrumented10mWT()

Examples

```
fs <- 100
t <- (0:(13 * fs - 1)) / fs
g <- function(t, mu, s) exp(-((t - mu) / s)^2)
yaw <- (pi / (0.6 * sqrt(pi))) * g(t, 5, 0.6) +
       (pi / (0.5 * sqrt(pi))) * g(t, 9.75, 0.5)
pitch <- 1.5 * g(t, 1, 0.3) + 1.3 * g(t, 11.5, 0.3)
av <- cbind(0.05 * sin(2 * pi * t), pitch, yaw)
tug <- instrumentedTUG(av, fs)
tug$total_duration
```

---

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

Description

Processes EMG at its native sampling rate, then aligns the resulting envelope (or MVC-normalized envelope) to MoCap sample times for feature analysis. Start times must be expressed on a shared clock. Alignment uses linear interpolation with the selected support policy; it does not estimate offsets. With `process = FALSE`, raw samples are interpolated without anti-alias filtering. With `outside = "NA"`, combined features remain missing outside raw-data support even if processing produces finite values beyond that support.

Usage

```
integrateEMGMoCap(
  mocap,
  emg,
  mocap_sampling_rate = NULL,
  emg_sampling_rate,
  mocap_assay = NULL,
  process = TRUE,
  ...,
  emg_start_time = 0,
  mocap_start_time = 0,
  outside = c("extend", "NA", "error")
)
```

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().

emg_start_time, mocap_start_time
: Finite scalar start times in seconds on the same reference clock. Defaults assume both recordings start at zero. These are known offsets; no synchronization or clock drift is estimated.

outside
: Handling of target times outside each channel's finite-data support: "extend" repeats endpoints (compatibility default), "NA" leaves unavailable times missing, and "error" rejects the alignment. Fewer than two finite observations produce missing output, or an error under "error". Internal gaps are still interpolated; this policy checks support boundaries, not internal gap duration. No anti-alias filter is added. Endpoint differences within floating-point roundoff are snapped to the endpoint (tolerance capped at one millionth of the shorter sample interval).

Value

A list with mocap, emg_aligned (aligned raw EMG, retained for compatibility), and combined data.frame (processed EMG when requested).

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 properties, using a recursive link-segment Newton-Euler chain (foot -> shank -> thigh).

Usage

```
inverseDynamics2D(
  joints,
  grf,
  sampling_rate,
  angles = NULL,
  angular_velocity = NULL,
  angular_acceleration = NULL,
  inertial = NULL,
  angle_unit = c("radian", "degree"),
  model = c("newton_euler", "quasi_static"),
  body_mass = NULL,
  body_height = NULL,
  gravity = .GRAVITY
)
```

Arguments

`joints`	Matrix/data.frame with columns `ankle_x`, `ankle_y`, `knee_x`, `knee_y`, `hip_x`, `hip_y`, and - for `model = "newton_euler"` - the foot distal end `toe_x`, `toe_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()`. Required for `model = "newton_euler"` unless `body_mass` is given; under `model = "quasi_static"` it only supplies an `I * alpha` correction and is ignored when omitted.
`angle_unit`	Unit of `angles`: `"radian"` or `"degree"`.
`model`	Which dynamics model to use. `"newton_euler"` (the default) runs the full recursive link-segment Newton-Euler chain via `inverseDynamicsRNE()`. `"quasi_static"` is the legacy massless-segment approximation, retained only for reproducing older results (see Details).

body_mass	Body mass in kg. Used to build the segment inertia table with estimateSegmentInertia() when inertial is not supplied.
body_height	Body height in m, passed to estimateSegmentInertia() when segment lengths must be estimated from stature. If NULL, segment lengths are measured from the marker data with scaleBodyModel().
gravity	Gravitational acceleration in m/s^2.

Details

The default "newton_euler" model propagates reactions distal to proximal: the ground reaction force and free moment enter at the foot, and every segment contributes its weight mg, its linear inertia ma_{com} and its angular inertia $I\alpha$. Segment centres of mass come from the anthropometric ratios in inertial, and their linear and angular accelerations from numerical differentiation of the marker trajectories, so a segment mass source (inertial or body_mass) and the foot distal end (toe_x, toe_y) are required.

Because the recursion differentiates the marker trajectories twice, it amplifies marker noise: the joint moments are only as good as the smoothing applied beforehand. On a static limb sampled at 200 Hz, 1 mm of white marker noise produces a spurious moment about 24 times the true value; a 6 Hz zero-lag low-pass (butterworthFilter(), or filterSignals() on the positions) reduces that by more than an order of magnitude. Filter the marker data before calling this function. The quasi-static model never differentiated positions, so this exposure is new.

"quasi_static" reproduces the pre-1.0 behaviour: each joint moment is the moment of the *ground reaction force alone* about that joint centre, plus an optional $I\alpha$ term. It omits segment weight and linear inertia entirely, so it returns identically zero whenever the limb is off the ground (the whole swing phase) and underestimates stance moments - at the hip by roughly 8% in quiet standing. It is kept only so that analyses published against the old implementation can be reproduced, and emits a message when selected.

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. model = "newton_euler" also returns the proximal joint reaction forces (ankle_fx, ankle_fy, ...).

References

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

See Also

inverseDynamicsRNE() for the underlying recursion, 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),
  toe_x   = rep(0.15, n), toe_y   = rep(0.01, 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, body_mass = 70)
```

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

Description

Computes ankle, knee, and hip net moment vectors in 3D from joint-center coordinates, GRF, COP, and segment inertial properties, using a recursive link-segment Newton-Euler chain (foot -> shank -> thigh).

Usage

```
inverseDynamics3D(
  joints,
  grf,
  sampling_rate,
  angles = NULL,
  angular_velocity = NULL,
  angular_acceleration = NULL,
  inertial = NULL,
  angle_unit = c("radian", "degree"),
  model = c("newton_euler", "quasi_static"),
  body_mass = NULL,
  body_height = NULL,
  gravity = .GRAVITY,
  vertical = c("y", "z")
)
```

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`, and - for `model = "newton_euler"` - the foot distal end `toe_x`, `toe_y`, `toe_z`.
`grf`	Matrix/data.frame with columns `fx`, `fy`, `fz` and optional `cop_x`, `cop_y`, `cop_z` (and, for `model = "newton_euler"`, optional free moments `tx`, `ty`, `tz`).
`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()`. Required for `model = "newton_euler"` unless `body_mass` is given; under `model = "quasi_static"` it only supplies an `I * alpha` correction and is ignored when omitted.

`angle_unit` Unit of angle-related inputs: `"radian"` or `"degree"`.

`model` Which dynamics model to use: `"newton_euler"` (the default, the full recursive chain via `inverseDynamicsRNE()`) or `"quasi_static"` (the legacy massless-segment approximation; see `inverseDynamics2D()`).

`body_mass` Body mass in kg, used to build the segment inertia table with `estimateSegmentInertia()` when `inertial` is not supplied.

`body_height` Body height in m, used when segment lengths must be estimated from stature rather than measured from the markers.

`gravity` Gravitational acceleration in m/s^2.

`vertical` Which coordinate axis points up, `"y"` (default) or `"z"`. Gravity acts along the negative of this axis, so it must match the laboratory convention of `joints`. The default follows this package's marker convention (x antero-posterior, y vertical, z medio-lateral) and the sagittal `inverseDynamics2D()`; note that force-plate hardware normally reports the vertical force as `fz`, so `grf` may need reordering to match. A mismatch is checked against the marker geometry and warned about, because the wrong axis silently removes every gravitational moment. Used only by `model = "newton_euler"`; the quasi-static model has no gravity term and therefore ignores it.

Details

See `inverseDynamics2D()` for what the two models compute and why the quasi-static one is retained only for reproducibility. The 3D moment balance uses an axisymmetric segment inertia and neglects the gyroscopic $\omega \times I\omega$ term and any spin about the segment long axis, which is not observable from two joint centres.

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. `model = "newton_euler"` also returns the proximal joint reaction force components (`ankle_fx`, `ankle_fy`, `ankle_fz`, ...).

References

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

See Also

`inverseDynamicsRNE()` for the underlying recursion, `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(0.05, n), ankle_z = rep(0.00, n),
  toe_x   = rep(0.15, n), toe_y   = rep(0.01, n), toe_z   = rep(0.00, n),
  knee_x = rep(0.00, n),  knee_y = rep(0.45, n),  knee_z = rep(0.00, n),
  hip_x = rep(0.00, n),   hip_y = rep(0.85, n),   hip_z = rep(0.00, n)
)
grf <- data.frame(
  fx = rep(50, n), fy = rep(700, n), fz = rep(0, n),
  cop_x = rep(0.02, n), cop_y = rep(0, n), cop_z = rep(0, n)
)
out <- inverseDynamics3D(joints, grf, sampling_rate = 100, body_mass = 70)
```

---

inverseDynamicsRNE *Recursive Newton-Euler inverse dynamics for the lower limb*

Description

Computes net ankle, knee and hip joint moments (and the joint reaction forces) with a proper distal-to-proximal recursive Newton-Euler chain (foot -> shank -> thigh). The ground reaction force and free moment enter at the foot; each segment contributes its inertia and gravity. Segment linear and angular accelerations are obtained by numerical differentiation of the marker-derived centre-of-mass and orientation trajectories.

Usage

```
inverseDynamicsRNE(
  joints,
  grf,
  inertia,
  sampling_rate,
  dims = c("2D", "3D"),
  gravity = .GRAVITY,
  vertical = c("y", "z")
)
```

Arguments

`joints`	A matrix/data.frame of joint-centre coordinates over time with columns `ankle_x`, `ankle_y`, `knee_x`, `knee_y`, `hip_x`, `hip_y` and the foot distal end `toe_x`, `toe_y` (for `dims = "3D"` add the `_z` columns).

grf	A matrix/data.frame with the ground reaction force fx, fy (and fz for 3D), the centre of pressure cop_x, cop_y (and cop_z), and an optional free moment: tz in 2D, or any of tx/ty/tz in 3D (each defaulting to 0).
inertia	A segment inertia table as returned by estimateSegmentInertia() (rows foot, shank, thigh).
sampling_rate	Sampling rate in Hz.
dims	"2D" (sagittal plane, default) or "3D".
gravity	Gravitational acceleration (default 9.80665 m/s^2).
vertical	Which coordinate axis points up: "y" (the default, matching the sagittal x-y convention) or "z". Only meaningful for dims = "3D"; a 2D chain must use "y". Gravity acts along the negative of this axis, so getting it wrong silently mis-signs every gravitational moment.

Value

A data.frame with time, the net joint moments (ankle_moment, knee_moment, hip_moment) and the proximal joint reaction forces for each joint.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. Wiley. Featherstone R (2008). "Rigid Body Dynamics Algorithms."

See Also

estimateSegmentInertia(), scaleBodyModel(), inverseDynamics2D()

Examples

```
n <- 50
joints <- data.frame(
  ankle_x = rep(0, n), ankle_y = rep(0.08, n),
  toe_x = rep(0.15, n), toe_y = rep(0.03, n),
  knee_x = rep(0, n),   knee_y = rep(0.48, n),
  hip_x = rep(0, n),    hip_y = rep(0.90, n))
grf <- data.frame(fx = rep(0, n), fy = rep(0, n),
                  cop_x = rep(0, n), cop_y = rep(0, n))
inertia <- estimateSegmentInertia(body_mass = 70, body_height = 1.75)
inverseDynamicsRNE(joints, grf, inertia, sampling_rate = 100)
```

---

inverseKinematicsMarkers

Model-based inverse kinematics from markers (planar)

Description

Solves the segment angles of a scaled planar chain that best fit observed marker positions, frame by frame, by least squares. Because the segment lengths are fixed by the model, the solution is consistent (unlike reading angles directly off noisy markers, which lets the limb stretch).

Usage

```
inverseKinematicsMarkers(markers, model, anchor_root = TRUE)
```

Arguments

markers	A [frame, node, 2] array of observed node positions (root and each joint/end), or a list of (n_segments + 1) x 2 matrices.
model	A planar_ik_model from scalePlanarModel().
anchor_root	If TRUE (default) the root is fixed to the observed root marker each frame; otherwise the root is also fitted.

Value

an ik_result: angles (frame x n_segments, absolute radians), rmse (per-frame marker RMS error, in position units), and n_frames.

References

Lu TW, O'Connor JJ (1999) global optimisation IK, J Biomech 32:129-134.

See Also

scalePlanarModel(), forwardKinematics2D()

Examples

```
model <- scalePlanarModel(rbind(c(0, 0), c(1, 0), c(2, 0)))
truth <- forwardKinematics2D(c(0.3, 0.7), model$lengths)
arr <- array(0, c(1, 3, 2)); arr[1, , ] <- truth
inverseKinematicsMarkers(arr, model)$angles        # ~ c(0.3, 0.7)
```

---

jointCoordinateSystem *Build embedded anatomical segment coordinate systems from marker clusters*

Description

Constructs a per-frame right-handed orthonormal coordinate system for each named body segment (e.g. pelvis, thigh/femur, shank/tibia, foot) from three markers: a proximal end, a distal end, and a lateral marker. The axes are X (antero-posterior), Y (long axis, pointing proximally) and Z (medio-lateral). These segment frames are the input to groodSuntayAngles and to the Grood-Suntay / ISB conventions of calculateJointAngles.

Usage

```
jointCoordinateSystem(pe, segments)
```

Arguments

pe	A PhysioExperiment with "position_x", "position_y" and "position_z" assays.
segments	A named list; each element is a list with proximal, distal and lateral marker names (columns of the position assays).

Value

A named list of n_frames x 3 x 3 arrays (one per segment); for each the three columns are the X (AP), Y (long) and Z (ML) unit axes.

References

Grood ES, Suntay WJ (1983). "A joint coordinate system for the clinical description of three-dimensional motions." J Biomech Eng, 105(2), 136-144. Wu G, et al. (2002, 2005). "ISB recommendation on definitions of joint coordinate systems." J Biomech.

See Also

groodSuntayAngles(), calculateJointAngles()

Examples

```
demo <- demoMoCapData(seed = 1)
jcs <- jointCoordinateSystem(demo$mocap,
  segments = list(shank_r = list(proximal = "Knee_R",
                                 distal = "Ankle_R", lateral = "Toe_R")))
names(jcs)
```

---

jointWork *Integrate joint power into concentric and eccentric work*

Description

Integrates a joint power time series (from computeJointPower()) with the trapezoidal rule to obtain concentric work (integral of positive "generation" power) and eccentric work (integral of negative "absorption" power). Because the trapezoidal rule is linear in the sampled values, concentric_work + eccentric_work equals the net integrated work to floating-point precision. Work can optionally be split per gait cycle or movement phase and normalised to body mass.

Usage

```
jointWork(power, sampling_rate, body_mass = NULL, windows = NULL)
```

Arguments

power	Numeric vector of joint power (W), or a data frame with a power column as returned by computeJointPower(..., split = TRUE).
sampling_rate	Sampling rate in Hz.
body_mass	Optional body mass in kilograms; if supplied, mass-normalised work columns (*_per_kg, J/kg) are added.
windows	Optional named list of integer index vectors, each selecting the samples of one gait cycle or phase (e.g. from segmentPhases()). Work is computed independently within each window. If NULL, work is computed over the whole series (window label "full").

Value

A data frame (class joint_work) with one row per window and columns window, concentric_work, eccentric_work, and net_work (J), plus the mass-normalised counterparts when body_mass is given.

References

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

See Also

computeJointPower(), labelPowerBursts().

Examples

```
t <- seq(0, 1, length.out = 101)
power <- 100 * sin(2 * pi * t)
jointWork(power, sampling_rate = 100, body_mass = 70)
```

---

kalmanOrientation *Kalman orientation filter (tilt + gyro-bias)*

Description

Fuses gyroscope and accelerometer with a per-axis linear Kalman filter that estimates both the tilt angle and the (slowly varying) gyroscope bias – the classic optimal complementary filter. Roll and pitch are corrected by gravity; yaw is integrated (or corrected by a magnetometer heading if supplied).

Usage

```
kalmanOrientation(
  gyro,
  accel,
  sampling_rate,
  mag = NULL,
  q_angle = 0.001,
  q_bias = 0.003,
  r_measure = 0.03
)
```

Arguments

gyro	n x 3 angular velocity (rad/s).
accel	n x 3 accelerometer.
sampling_rate	Sampling rate (Hz).
mag	Optional n x 3 magnetometer for yaw.
q_angle, q_bias	Process-noise variances for angle and bias (defaults 1e-3, 3e-3).
r_measure	Measurement-noise variance of the accelerometer angle (default 0.03).

Value

a data frame time, roll, pitch, yaw (rad) and quaternion columns, plus attribute bias (estimated gyro bias per axis).

References

Lauszus K (2012) Kalman IMU; Brown & Hwang (1997).

See Also

mahonyAHRS(), estimateOrientation()

Examples

```
n <- 300; sr <- 100; t <- seq_len(n) / sr
roll <- 0.4 * sin(2 * pi * 0.4 * t); pitch <- 0.3 * sin(2 * pi * 0.25 * t)
accel <- cbind(-sin(pitch), sin(roll) * cos(pitch), cos(roll) * cos(pitch))
gyro <- cbind(c(0, diff(roll)) * sr + 0.05, c(0, diff(pitch)) * sr - 0.03, 0)
est <- kalmanOrientation(gyro, accel, sampling_rate = sr)
max(abs(est$roll - roll))
```

`labelPowerBursts` *Label joint power bursts with Winter nomenclature*

Description

Segments a single-gait-cycle joint power time series into contiguous same-sign bursts and assigns Winter's power-phase labels: `A1/A2` for the ankle, `H1-H3` for the hip, and `K1-K4` for the knee. Each canonical burst has an expected sign (generation vs absorption) and an approximate location within the gait cycle; detected bursts are matched to the nearest unused canonical burst of the same sign.

Usage

```
labelPowerBursts(
  power,
  sampling_rate,
  joint = c("ankle", "knee", "hip"),
  body_mass = NULL
)
```

Arguments

`power`	Numeric vector of joint power (W) over one gait cycle, or a data frame with a `power` column.
`sampling_rate`	Sampling rate in Hz.
`joint`	One of `"ankle"`, `"knee"`, or `"hip"`.
`body_mass`	Optional body mass in kilograms; if supplied, a `work_per_kg` column (J/kg) is added.

Details

The `power` vector is assumed to span exactly one gait cycle (initial contact to the next ipsilateral initial contact), so burst timing is reported as a percentage of the cycle.

Value

A data frame (class `power_bursts`) with one row per detected burst: `label` (canonical burst name or `NA` if unmatched), `joint`, `type` (`"generation"` or `"absorption"`), `start_pct`, `end_pct`, `peak_power` (W, signed), and `work` (J), ordered by time.

References

Winter DA (2009). "Biomechanics and Motor Control of Human Movement." 4th ed. John Wiley & Sons. Power-phase nomenclature A1-A2 / H1-H3 / K1-K4.

See Also

`computeJointPower()`, `jointWork()`.

Examples

```
t <- seq(0, 1, length.out = 101)
ankle_power <- ifelse(t < 0.4, -50 * sin(pi * t / 0.4),
                      120 * sin(pi * (t - 0.4) / 0.2) * (t < 0.6))
labelPowerBursts(ankle_power, sampling_rate = 100, joint = "ankle")
```

---

ldlj *Log Dimensionless Jerk (LDLJ) movement smoothness*

Description

The log of the (speed-based) dimensionless jerk, negated so that more negative = less smooth.

Usage

```
ldlj(speed, fs)
```

Arguments

speed	Numeric speed profile.
fs	Sampling frequency in Hz.

Value

The LDLJ (negative scalar).

References

Balasubramanian et al. (2015); Melendez-Calderon et al. (2021).

Examples

```
t <- seq(0, 1, length.out = 200); tau <- t
speed <- 30 * tau^2 - 60 * tau^3 + 30 * tau^4
ldlj(speed, fs = 200)
```

limbAtaxiaIndex *Limb-ataxia composite from reaching sub-metrics*

Description

Aggregates the standard limb-ataxia sub-metrics, each oriented so that a higher value means more ataxic movement. Supply them from the existing primitives: `dysmetria` from `endpointError()` (`absolute_error` or the magnitude of `constant_error`), `decomposition` from `movementUnits()` (the submovement count), `smoothness` as a jerk/roughness score where higher is worse (e.g. `-sparc` from `sparc()`, since SPARC is more negative for less smooth movement, or `ldlj` magnitude), and `irregularity` from `pathStraightness()` (`index_of_curvature`).

Usage

```
limbAtaxiaIndex(
  dysmetria = NA_real_,
  decomposition = NA_real_,
  smoothness = NA_real_,
  irregularity = NA_real_,
  reference = NULL,
  weights = NULL
)
```

Arguments

`dysmetria`, `decomposition`, `smoothness`, `irregularity`	Numeric sub-metrics (higher = more ataxic); any may be `NA` to omit it.
`reference`	Optional data.frame of the same-named columns measured in a healthy reference sample; when given, each sub-metric is z-standardised against it and averaged into `composite`. Without a reference the composite is `NA` and only the sub-metrics are returned.
`weights`	Optional named weights for the composite (default equal).

Value

An S3 `limb_ataxia` list: `metrics`, per-metric `z`, and `composite`.

See Also

`pathStraightness()`, `reachingKinematics()`, `gaitAtaxiaIndex()`

Examples

```
limbAtaxiaIndex(dysmetria = 3.2, decomposition = 4, smoothness = 5.1,
                irregularity = 1.4)
```

`limitsOfStability` *Limits of Stability (LOS) reaction time, excursion and directional control*

Description

Quantifies a single limits-of-stability leaning trial toward a target direction from a CoP trace: reaction time, endpoint and maximum excursion (optionally as a percentage of a theoretical limit), and directional control.

Usage

```
limitsOfStability(
  cop,
  sampling_rate,
  target,
  ap = NULL,
  ml = NULL,
  onset_cue = 1L,
  onset_frac = 0.05,
  los_distance = NULL
)
```

Arguments

`cop`, `ap`, `ml`	CoP trace, as in `swayMetrics()` (a single leaning trial).
`sampling_rate`	Sampling rate in Hz.
`target`	Target direction: an angle in degrees (0 = +AP/forward, 90 = +ML/right, measured counter-clockwise from forward) or a length-2 numeric `c(ap, ml)` direction vector.
`onset_cue`	Sample index of the go cue (default 1).
`onset_frac`	Fraction of the maximum excursion defining movement onset (default 0.05).
`los_distance`	Optional theoretical limit-of-stability distance (CoP units) for the target; if supplied, excursions are also returned as a percentage of it.

Details

The CoP is projected onto the unit vector toward the target. Reaction time is the delay from the go cue to movement onset (projection first exceeding `onset_frac` of the maximum excursion). Endpoint excursion is the projection at the end of the initial movement (its first local maximum after onset); maximum excursion is the largest projection. Directional control is $100\times$ the path travelled toward the target divided by the total path length.

Value

A `los_result` object with `reaction_time`, `endpoint_excursion`, `max_excursion`, optional `endpoint_pct` / `max_pct`, and `directional_control`.

References

Nashner LM (1997). "Computerized dynamic posturography."

See Also

sensoryOrganizationTest(), swayMetrics()

Examples

```
t <- seq(0, 3, by = 0.01)
lean <- pmin(t / 1.5, 1) * 5             # ramp forward to 5 units
cop <- data.frame(cop_x = rnorm(length(t), 0, 0.02), cop_y = lean)
limitsOfStability(cop, sampling_rate = 100, target = 0)
```

---

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)
```

`localDynamicStability` *Local dynamic stability (short- and long-term divergence)*

Description

Local dynamic stability quantifies how a system responds to small natural perturbations, as the exponential divergence rate of neighbouring trajectories over the short term (0-1 stride, `lambda_short`) and long term (4-10 strides, `lambda_long`), following Dingwell & Cusumano (2000). Higher divergence means less stable locomotion.

Usage

```
localDynamicStability(
  x,
  stride_samples,
  delay = NULL,
  dim = NULL,
  sampling_rate = 1
)
```

Arguments

`x`	Numeric time series (e.g. trunk acceleration or a joint angle).
`stride_samples`	Samples per stride (sets the divergence windows).
`delay`, `dim`	Embedding parameters; `NULL` estimates them.
`sampling_rate`	Sampling rate in Hz.

Value

A `local_dynamic_stability` object with `lambda_short`, `lambda_long` (per stride), the `divergence` curve, and embedding info.

References

Dingwell JB, Cusumano JJ (2000). Chaos 10(4):848-863.

See Also

`maxLyapunovExponent()`

Examples

```
x <- sin(seq(0, 20 * pi, length.out = 600))
lds <- localDynamicStability(x, stride_samples = 60)
lds$lambda_short
```

mahonyAHRS *Mahony AHRS orientation filter*

Description

Estimates orientation by integrating the gyroscope while correcting drift with the accelerometer (gravity) and optional magnetometer through a proportional+integral complementary feedback on SO(3) (Mahony et al. 2008).

Usage

```
mahonyAHRS(gyro, accel, sampling_rate, kp = 1, ki = 0.1)
```

Arguments

gyro	n x 3 angular velocity (rad/s), columns x, y, z.
accel	n x 3 accelerometer (any consistent unit; normalised internally).
sampling_rate	Sampling rate (Hz).
kp, ki	Proportional and integral feedback gains (default 1, 0.1).

Value

a data frame time, roll, pitch, yaw (rad) and q_w, q_x, q_y, q_z, matching estimateOrientation().

References

Mahony R, et al. (2008) IEEE Trans Autom Control 53:1203-1218.

See Also

kalmanOrientation(), estimateOrientation()

Examples

```
n <- 200; sr <- 100; t <- seq_len(n) / sr
roll <- 0.3 * sin(2 * pi * 0.5 * t); pitch <- 0.2 * cos(2 * pi * 0.3 * t)
accel <- cbind(-sin(pitch), sin(roll) * cos(pitch), cos(roll) * cos(pitch))
gyro <- cbind(c(0, diff(roll)) * sr, c(0, diff(pitch)) * sr, 0)
est <- mahonyAHRS(gyro, accel, sampling_rate = sr)
max(abs(est$roll - roll))
```

`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
)
```

`maxLyapunovExponent` *Largest Lyapunov exponent (Rosenstein)*

Description

Estimates the largest Lyapunov exponent from the average logarithmic divergence of initially nearby trajectories in a delay-embedded phase space (Rosenstein et al. 1993). A positive exponent indicates sensitive dependence on initial conditions (chaos / local instability).

Usage

```
maxLyapunovExponent(
  x,
  delay = NULL,
  dim = NULL,
  sampling_rate = 1,
  mean_period = NULL,
  max_steps = NULL,
  fit_range = NULL
)
```

Arguments

`x`	Numeric time series.
`delay`, `dim`	Embedding delay and dimension; `NULL` estimates them via `timeDelayEmbed()`.
`sampling_rate`	Sampling rate in Hz (scales the exponent to per-second).
`mean_period`	Theiler window in samples excluding temporally-close neighbours; `NULL` estimates it from the mean signal period.
`max_steps`	Number of forward steps to track divergence; `NULL` uses a default derived from the series length.
`fit_range`	Integer vector `c(from, to)` (in steps) of the divergence curve to fit the slope; `NULL` uses an early near-linear window.

Value

A `lyapunov_exponent` object with `lambda` (per second), the `divergence` curve, `fit_range`, `delay` and `dim`.

References

Rosenstein MT, Collins JJ, De Luca CJ (1993). Physica D 65:117-134.

See Also

`localDynamicStability()`, `timeDelayEmbed()`

Examples

```
x <- sin(seq(0, 20 * pi, length.out = 600))
mle <- maxLyapunovExponent(x)
mle$lambda
```

`mCTSIB` *Modified Clinical Test of Sensory Interaction on Balance (mCTSIB)*

Description

Summarises sway velocity (or any sway index) across the four mCTSIB sensory conditions (eyes open / closed on a firm surface and on foam) and returns the per-condition means and the composite (mean of the four condition means).

Usage

```
mCTSIB(
  eyes_open_firm = NULL,
  eyes_closed_firm = NULL,
  eyes_open_foam = NULL,
  eyes_closed_foam = NULL,
  conditions = NULL
)
```

Arguments

`eyes_open_firm`, `eyes_closed_firm`, `eyes_open_foam`, `eyes_closed_foam`
: Numeric vectors of per-trial sway velocities (or another sway index) for each condition. Missing conditions may be `NULL`.

`conditions`
: Alternatively, a named list with those four elements.

Value

An `mctsib_result` object with `condition_means`, `condition_sd`, `n_trials` and the `composite` mean.

References

Cohen H, Blatchly CA, Gombash LL (1993). Phys Ther 73(6):346-351.

See Also

`sensoryOrganizationTest()`, `swayMetrics()`

Examples

```
mCTSIB(eyes_open_firm = c(0.4, 0.5),
       eyes_closed_firm = c(0.6, 0.7),
       eyes_open_foam = c(0.9, 1.0),
       eyes_closed_foam = c(1.6, 1.8))
```

`movementAnalysisProfile` *Movement Analysis Profile (GVS per variable plus the GPS)*

Description

The Movement Analysis Profile (Baker et al. 2009) collects the per-variable Gait Variable Scores together with the overall Gait Profile Score.

Usage

```
movementAnalysisProfile(kinematics, norm = NULL)
```

Arguments

`kinematics`	Subject kinematics: a numeric matrix with one row per kinematic variable (row names matching `norm$variables`) and one column per gait-cycle point, or a named list of per-variable curves. Curves are time-normalised to the norm's cycle length via `normalizeMovement()`.
`norm`	A `gait_norm` normative reference (from `PhysioGaitNorm::loadGaitNorm()`); if `NULL`, the default reference is loaded from PhysioGaitNorm.

Value

A `movement_analysis_profile` object: a list with `gvs` (named numeric vector), `gps` (numeric), `variables`, and `cycle_length`.

References

Baker R, et al. (2009). Gait & Posture 30(3):265-269.

See Also

`gaitProfileScore()`, `gaitVariableScore()`, `plotMAP()`

Examples

```
norm <- list(variables = c("a", "b"),
             mean = matrix(0, 2, 51, dimnames = list(c("a", "b"), NULL)),
             cycle_length = 51)
movementAnalysisProfile(matrix(1, 2, 51, dimnames = list(c("a", "b"), NULL)),
                        norm)
```

`movementSmoothness` *Movement-smoothness battery (SPARC + LDLJ)*

Description

Computes the SPARC and log dimensionless jerk smoothness metrics for a speed profile. Accepts either a numeric speed vector (with `fs`) or a `PhysioExperiment`, in which case the speed profile is pulled from the `assay` (computing it with `computeVelocity` / `computeSpeed` when absent) and the sampling rate from the object. When several markers are present each is scored.

Usage

```
movementSmoothness(x, fs = NULL, assay = "speed", marker = NULL, ...)
```

Arguments

`x`	A numeric speed profile, or a `PhysioExperiment`.
`fs`	Sampling frequency in Hz (ignored / taken from the object when `x` is a `PhysioExperiment`).
`assay`	Speed assay to use for a `PhysioExperiment` (default `"speed"`).
`marker`	Optional marker (column) name or index to score; by default all markers of the speed assay are scored.
`...`	Passed to `sparc`.

Value

An S3 `"movement_smoothness"` object: a list with `sparc` and `ldlj` (named by marker when a `PhysioExperiment` is scored), `dimensionless_jerk`, `fs` and `n`.

References

Balasubramanian et al. (2015); Melendez-Calderon et al. (2021).

Examples

```
t <- seq(0, 1, length.out = 200); tau <- t
speed <- 30 * tau^2 - 60 * tau^3 + 30 * tau^4
movementSmoothness(speed, fs = 200)
```

movementTime *Reaching movement time*

Description

Reaching movement time

Usage

```
movementTime(
  speed,
  fs,
  onset_threshold = 0.05,
  threshold_type = c("relative", "absolute")
)
```

Arguments

speed	Numeric non-negative tangential-speed profile.
fs	Sampling frequency in Hz.
onset_threshold	Non-negative threshold, interpreted according to threshold_type.
threshold_type	"relative" for a fraction of peak speed or "absolute" for speed units.

Value

Movement duration in seconds. The 1-based onset and offset samples are stored as attributes. Returns NA when no sample exceeds threshold.

Examples

```
speed <- c(rep(0, 20), seq(0, 1, length.out = 30),
           seq(1, 0, length.out = 30), rep(0, 20))
movementTime(speed, fs = 100)
```

movementUnits *Count reaching movement units*

Description

Counts accepted speed peaks as movement units (submovements). A peak must exceed both a relative height threshold and a relative prominence threshold.

Usage

```
movementUnits(
  speed,
  fs = NULL,
  height_frac = 0.05,
  prominence_frac = 0.1,
  min_peak_distance = NULL
)
```

Arguments

speed	Numeric non-negative tangential-speed profile.
fs	Optional sampling frequency in Hz. Required only when min_peak_distance is supplied.
height_frac	Peak-height threshold as a fraction of maximum speed.
prominence_frac	Minimum peak prominence as a fraction of maximum speed.
min_peak_distance	Optional minimum inter-peak interval in seconds.

Value

Integer movement-unit count. Accepted 1-based peak indices are stored in the "peaks" attribute.

References

Rohrer B, Fasoli S, Krebs HI, et al. (2002). Movement smoothness changes during stroke recovery. *Journal of Neuroscience*, 22:8297-8304.

Examples

```
t <- seq(0, 1, length.out = 500)
speed <- exp(-((t - 0.3) / 0.04)^2) + exp(-((t - 0.7) / 0.04)^2)
movementUnits(speed)
```

---

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)
```

---

nhptDexterity *Instrumented Nine-Hole-Peg dexterity from a hand-speed profile*

Description

Turns the hand-speed profile of a peg-transport task (the Nine-Hole-Peg Test of fine hand use) into instrumented dexterity metrics: the number of detected peg transports (speed peaks), the transport rate, the mean and coefficient of variation of the inter-transport intervals (movement consistency) and the overall smoothness. This is the sensor-derived complement to the clinical timed NHPT score, and realises ICF d440 (fine hand use).

Usage

```
nhptDexterity(speed, sampling_rate, ...)
```

Arguments

speed	Numeric hand speed profile over the whole task.
sampling_rate	Sampling rate in Hz.
...	Passed to movementUnits() (peak height/prominence controls).

Value

an nhpt_dexterity list: icf_code, n_transports, total_time, transport_rate (transports/s), mean_interval, cv_interval (interval consistency) and smoothness (SPARC).

References

Mathiowetz V, et al. (1985) Am J Occup Ther 39:386-391 (NHPT).

See Also

movementUnits(), adlReachTask()

Examples

```
fs <- 100
bell <- function(n) { u <- seq(0, 1, length.out = n); 30 * u^2 * (1 - u)^2 }
# nine evenly spaced transports
v <- unlist(replicate(9, c(bell(40), numeric(20)), simplify = FALSE))
nhptDexterity(v, fs)$n_transports
```

---

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)
```

---

pathStraightness *Movement-path straightness*

Description

The straightness of a movement trajectory: the ratio of the straight-line (direct) distance between the first and last points to the travelled path length. A straight reach scores 1; a curved or decomposed path scores below 1. The reciprocal (index of curvature) grows with path irregularity, a hallmark of limb ataxia.

Usage

```
pathStraightness(trajectory)
```

Arguments

trajectory	Numeric matrix of positions over time (rows = samples, columns = 1-3 spatial axes), or a numeric vector for a single axis.

Value

A list: path_length, direct_distance, straightness (direct / path, in (0, 1]) and index_of_curvature (path / direct, >= 1).

See Also

reachingKinematics(), limbAtaxiaIndex()

Examples

```
pathStraightness(cbind(c(0, 0, 1), c(0, 1, 1)))$straightness   # L-path
```

peakPressure *Calculate a peak plantar-pressure map*

Description

Calculate a peak plantar-pressure map

Usage

```
peakPressure(pm, contact_threshold = 0)
```

Arguments

pm	A pressure_movie.
contact_threshold	Pressure threshold in kPa. Cells must exceed this threshold to count toward contact area and mean pressure.

Value

A numeric peak-pressure matrix in kPa. Attributes contain the global peak, peak cell and frame, contact area, and mean active pressure.

References

Rosenbaum D, Becker H-P (1997). Plantar pressure distribution measurements. *Foot and Ankle Surgery*, 3:1-14.

Examples

```
pm <- pressureMovie(array(1:12, c(2, 2, 3)), 100)
peakPressure(pm)
```

peakTorque *Peak torque from isometric or isokinetic dynamometry*

Description

Finds the maximum gravity-corrected torque. For isokinetic trials with an angular-velocity trace, the search is restricted to samples within velocity_tol of the target velocity so acceleration and deceleration transients do not determine the peak.

Usage

```
peakTorque(
  torque,
  sampling_rate,
  mode = c("isokinetic", "isometric"),
  angle = NULL,
  angular_velocity = NULL,
  target_velocity = NULL,
  velocity_tol = 0.1,
  at_angle = NULL,
  gravity = NULL
)
```

Arguments

`torque`	Numeric torque trace in N m.
`sampling_rate`	Sampling rate in Hz.
`mode`	Either `"isokinetic"` or `"isometric"`.
`angle`	Optional joint-angle trace in degrees.
`angular_velocity`	Optional angular-velocity trace in degrees/s.
`target_velocity`	Optional target angular velocity in degrees/s. The largest observed absolute velocity is used when omitted.
`velocity_tol`	Non-negative relative tolerance around target velocity.
`at_angle`	Optional angles in degrees at which torque is interpolated along the rising angular limb.
`gravity`	Optional scalar or sample-wise gravity torque in N m to subtract before analysis.

Value

A `peak_torque` object containing the peak value, time, sample, angle, angle-specific values, mode, and searched sample indices.

References

Aagaard P, Simonsen EB, Andersen JL, Magnusson P, Dyhre-Poulsen P (2002). Increased rate of force development and neural drive of human skeletal muscle following resistance training. *Journal of Applied Physiology*, 93:1318-1326. doi:10.1152/japplphysiol.00283.2002

Examples

```
torque <- c(seq(0, 120, length.out = 101), rep(120, 50))
peakTorque(torque, sampling_rate = 100, mode = "isometric")
```

peakVelocity *Peak reaching velocity*

Description

Peak reaching velocity

Usage

```
peakVelocity(speed)
```

Arguments

speed	Numeric non-negative tangential-speed profile.

Value

Peak speed. Its first 1-based sample index is stored in the "index" attribute. Returns NA when all samples are missing.

Examples

```
peakVelocity(c(0, 1, 3, 2, 0))
```

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()

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phaseDurations(segmentPhases(data, events, schema_gait))
```

---

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()

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phaseRatios(segmentPhases(data, events, schema_gait))
```

---

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()

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phaseTiming(segmentPhases(data, events, schema_gait))
```

---

pipelineACLrts *ACL return-to-sport pipeline (limb symmetry index)*

Description

Computes the Limb Symmetry Index (LSI) for a battery of functional tests and evaluates the return-to-sport (RTS) criterion that every LSI meets a threshold (default 90\ tests where a higher score is better, and inverted otherwise.

Usage

```
pipelineACLrts(tests, threshold = 90)
```

Arguments

`tests`	A data frame with columns `test`, `involved`, `uninvolved`, and optionally `higher_better` (logical, default `TRUE`); or a named list of `c(involved, uninvolved)` pairs.
`threshold`	LSI percentage required to pass each test (default 90).

Value

An `acl_rts_report` object with a per-test `lsi` data frame and the overall `rts_ready` flag.

References

Grindem H, et al. (2016). Br J Sports Med 50(13):804-808.

Examples

```
tests <- list(hop = list(involved = 90, uninvolved = 100))
pipelineACLrts(tests)
```

`pipelineAmputee` *Amputee prosthetic-side work-asymmetry pipeline*

Description

Compares the positive (concentric) joint work generated by the prosthetic and intact limbs (via `jointWork()`) and reports a work-asymmetry index. A value near 0 is symmetric; positive values indicate the intact limb generates more positive work than the prosthetic limb.

Usage

```
pipelineAmputee(
  power_prosthetic,
  power_intact,
  sampling_rate,
  body_mass = NULL
)
```

Arguments

`power_prosthetic`, `power_intact`	Joint power time series (W or W/kg) for the prosthetic and intact limbs.
`sampling_rate`	Sampling rate in Hz.
`body_mass`	Optional body mass (kg) for mass-normalised work.

Value

An `amputee_report` object.

References

Winter DA (2009); Nolan & Lees (2000).

See Also

`jointWork()`

Examples

```
set.seed(1)
power_prosthetic <- abs(sin(seq(0, 10 * pi, length.out = 1000))) * 2
power_intact <- abs(sin(seq(0, 10 * pi, length.out = 1000))) * 3
pipelineAmputee(power_prosthetic, power_intact, sampling_rate = 100)
```

pipelineCPgait *Cerebral-palsy gait pipeline (GDI + pathology flags)*

Description

Computes the Gait Deviation Index (Schwartz & Rozumalski 2008) for a subject's kinematics, categorises it, adds the kinematic pathology flags detectable from the same waveforms (stiff knee, Trendelenburg), and records the GMFCS level for stratified interpretation.

Usage

```
pipelineCPgait(kinematics, norm = NULL, gmfcs = NULL)
```

Arguments

kinematics	Subject kinematics passed to gaitDeviationIndex() (a variables x cycle-points matrix with row names matching the norm).
norm	A gait_norm reference (from PhysioGaitNorm::loadGaitNorm()); if NULL, the default is loaded from PhysioGaitNorm.
gmfcs	Optional Gross Motor Function Classification System level (I-V).

Value

A cp_gait_report object.

References

Schwartz & Rozumalski (2008); Palisano et al. (1997) GMFCS.

See Also

gaitDeviationIndex(), classifyGaitPatterns()

Examples

```
if (requireNamespace("PhysioGaitNorm", quietly = TRUE)) {
  set.seed(1)
  vars <- c("pelvic_tilt", "pelvic_obliquity", "pelvic_rotation",
            "hip_flexion", "hip_adduction", "hip_rotation",
            "knee_flexion", "ankle_dorsiflexion", "foot_progression")
  kinematics <- matrix(rnorm(9 * 51), 9, 51, dimnames = list(vars, NULL))
  pipelineCPgait(kinematics)
}
```

`pipelinePDfog` *Parkinson's freezing-of-gait pipeline (freeze index)*

Description

Detects freezing of gait (FOG) from a windowed spectral freeze index: the ratio of power in the freeze band (default 3-8 Hz) to power in the locomotor band (default 0.5-3 Hz) of an accelerometer signal (Moore et al. 2008; Bachlin et al. 2010). A window is flagged as FOG when the freeze index exceeds `fi_threshold` and the band power exceeds `power_threshold` (excluding rest).

Usage

```
pipelinePDfog(
  accel,
  sampling_rate,
  window_sec = 4,
  step_sec = 0.5,
  freeze_band = c(3, 8),
  locomotor_band = c(0.5, 3),
  fi_threshold = 2,
  power_threshold = NULL
)
```

Arguments

`accel`	Accelerometer signal (e.g. shank/trunk vertical or resultant).
`sampling_rate`	Sampling rate in Hz.
`window_sec, step_sec`	Sliding-window length and step (seconds).
`freeze_band, locomotor_band`	Frequency bands (Hz).
`fi_threshold`	Freeze-index threshold for FOG (default 2).
`power_threshold`	Minimum band power for a window to be considered (excludes rest). `NULL` uses the 10th percentile of windowed band power as a quiet-baseline floor (robust to a loud walking bout, unlike a maximum-relative gate); pass a fixed absolute value if the sensor units are known.

Value

A `pd_fog_report` object with a per-window `windows` data frame (`time`, `freeze_index`, `power`, `fog`) and the overall `fog_fraction`.

References

Moore ST, et al. (2008); Bachlin M, et al. (2010).

Examples

```
set.seed(1)
accel <- matrix(rnorm(2000 * 3), 2000, 3)
pipelinePDfog(accel, sampling_rate = 100)
```

---

pipelineStrokePropulsion

Stroke paretic-propulsion pipeline

Description

Quantifies propulsion symmetry after stroke from the anterior-posterior ground reaction force of each limb: the paretic propulsion `Pp` is the paretic propulsive (anterior) impulse divided by the total propulsive impulse of both limbs (Bowden et al. 2006). A symmetric gait gives `Pp = 0.5`; reduced paretic output gives `Pp < 0.5`.

Usage

```
pipelineStrokePropulsion(ap_paretic, ap_nonparetic, sampling_rate)
```

Arguments

`ap_paretic`, `ap_nonparetic`	Anterior-posterior ground reaction force of the paretic and non-paretic limbs over the analysis window (anterior / propulsive positive).
`sampling_rate`	Sampling rate in Hz.

Value

A `stroke_propulsion_report` object.

References

Bowden MG, et al. (2006). Stroke 37(3):872-876.

Examples

```
set.seed(1)
ap_paretic <- sin(seq(0, 10 * pi, length.out = 1000))
ap_nonparetic <- sin(seq(0, 10 * pi, length.out = 1000)) * 1.3
pipelineStrokePropulsion(ap_paretic, ap_nonparetic, sampling_rate = 100)
```

`planarLimbModel` *Planar two-segment limb model (equations of motion)*

Description

Builds the analytic dynamics of a planar two-link limb with distributed segment mass: the mass matrix `M(q)`, the Coriolis/centrifugal matrix, the gravity vector, and forward/inverse dynamics. Angles are joint angles (proximal, then distal relative to the proximal segment), measured so gravity acts in `-y`.

Usage

```
planarLimbModel(
  mass,
  length,
  com = length/2,
  inertia = mass * length^2/12,
  g = 9.81
)
```

Arguments

`mass`	Length-2 segment masses (kg).
`length`	Length-2 segment lengths (m).
`com`	Length-2 distances of each segment centre of mass from its proximal joint (m); default `length / 2`.
`inertia`	Length-2 segment moments of inertia about the CoM (kg m^2); default the thin-rod `mass * length^2 / 12`.
`g`	Gravitational acceleration (m/s^2, default 9.81; use 0 for a horizontal-plane / no-gravity model).

Value

a `planar_limb_model`: functions `M(q)`, `coriolis(q, qd)` (matrix), `gravity(q)`, `forward_accel(q, qd, tau)`, `inverse_dynamics(q, qd, qdd)`, `energy(q, qd)` (kinetic, potential, total), and `ndof = 2`.

References

Spong MW, et al. (2006) Robot Modeling and Control.

See Also

`forwardDynamics()`, `predictiveSimulation()`

Examples

```
m <- planarLimbModel(mass = c(7, 3), length = c(0.4, 0.4))
m$energy(c(0, 0), c(1, 0))$total
```

`plotBlandAltman` *Bland-Altman agreement plot*

Description

Draws the Bland-Altman difference-vs-mean scatter that `blandAltman` only computes: the bias (mean difference), the upper/lower limits of agreement (LoA), and shaded confidence bands for the bias and each LoA. The numerics come from `blandAltman()` so the plotted reference lines are single-sourced and cannot drift from the computed statistics.

Usage

```
plotBlandAltman(
  x,
  y,
  confidence = 0.95,
  proportional_bias = FALSE,
  units = NULL,
  colorblind = TRUE
)
```

Arguments

`x, y`	Numeric vectors of paired measurements (two methods, or two time points). Must be the same length with at least 2 pairs (validated by `blandAltman()`).
`confidence`	Confidence level for the limits of agreement and the shaded confidence bands (default 0.95).
`proportional_bias`	Logical. If `TRUE`, model the bias and LoA as linear in the mean (Bland-Altman regression): the bias line is the OLS fit of the difference on the mean with a shaded fitted-line confidence band, and the LoA are the fit $\pm z \cdot s_{resid}$. Use when the difference scales with magnitude. Requires at least 3 pairs (the regression residual variance is undefined at n = 2). Default `FALSE` (constant bias/LoA).
`units`	Optional measurement unit appended to the axis labels.
`colorblind`	Logical; if `TRUE` (default) use the ecosystem colorblind-safe palette (`physioPalette`).

Value

A `ggplot` object.

References

Bland JM, Altman DG (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *Lancet*, 327(8476), 307-310. doi:10.1016/S01406736(86)908378

See Also

`PhysioExperiment::blandAltman()` for the underlying statistics.

Examples

```
set.seed(1)
x <- rnorm(50, 10, 2)
y <- x + rnorm(50, 0, 0.5)
plotBlandAltman(x, y)
```

---

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()

Examples

```
x <- sin(seq(0, pi, length.out = 100)) * 30
y <- sin(seq(0, pi, length.out = 100) + 0.3) * 30
dr <- dtwDistance(x, y)
plotDTW(dr)
```

`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.

Examples

```
set.seed(1)
curves <- matrix(rep(sin(seq(0, pi, length.out = 101)), 20) +
                   rnorm(101 * 20, 0, 0.1), 101, 20)
fp <- fPCA(curves)
plotFPCA(fp, type = "variance")
```

`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.

Examples

```
set.seed(1)
waveforms <- matrix(sin(seq(0, pi, length.out = 101)) + rnorm(101 * 8, 0, 0.2),
                    101, 8)
plotGroupComparison(waveforms, groups = rep(c("A", "B"), 4))
```

---

plotMAP *Plot a Movement Analysis Profile*

Description

Draws the Movement Analysis Profile (Baker et al. 2009) as a bar chart of the Gait Variable Score for each kinematic variable, with a reference line at the overall Gait Profile Score.

Usage

```
plotMAP(map, title = "Movement Analysis Profile")
```

Arguments

map	A movement_analysis_profile object from movementAnalysisProfile().
title	Plot title.

Value

A ggplot object.

See Also

movementAnalysisProfile()

Examples

```
norm <- list(variables = c("a", "b"),
             mean = matrix(0, 2, 51, dimnames = list(c("a", "b"), NULL)),
             cycle_length = 51)
map <- movementAnalysisProfile(
  matrix(1, 2, 51, dimnames = list(c("a", "b"), NULL)), norm)
plotMAP(map)
```

---

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.

Examples

```
set.seed(1)
waveforms <- matrix(rnorm(101 * 4), 101, 4)
colnames(waveforms) <- paste0("ch", 1:4)
plotMultiPanel(waveforms)
```

---

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()

Examples

```
set.seed(1)
d <- matrix(rnorm(1000), 100, 10)
pca <- waveformPCA(d, method = "features")
plotPCAScatter(pca)
```

---

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()

Examples

```
set.seed(1)
d <- matrix(rnorm(1000), 100, 10)
pca <- waveformPCA(d, method = "features")
plotPCAVariance(pca)
```

`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.

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phases <- segmentPhases(data, events, schema_gait)
plotPhaseDurations(list(phases))
```

---

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)
```

---

plotPressureMap *Plot a plantar-pressure map*

Description

Renders peak, mean, pressure-time-integral, or individual-frame pressure. For `type = "frame"`, omitting `frame` selects the peak-total-pressure frame. Supplying `n_facets` explicitly while omitting `frame` instead displays peak-pressure maps over equal stance windows.

Usage

```
plotPressureMap(
  pm,
  type = c("peak", "mean", "pti", "frame"),
  frame = NULL,
  n_facets = 6L,
  contact_threshold = 0,
  palette = "viridis",
  flip_ap = FALSE
)
```

Arguments

`pm`	A `pressure_movie`.
`type`	Map aggregation: `"peak"`, `"mean"`, `"pti"`, or `"frame"`.
`frame`	Frame index for `type = "frame"`.
`n_facets`	Number of stance windows when explicitly supplied for a faceted frame plot.
`contact_threshold`	Values at or below this pressure are not drawn.

palette	Viridis palette name/option, a single high-end colour, or a vector of gradient colours.
flip_ap	Logical; reverse the displayed anteroposterior axis.

Value

A ggplot object.

Examples

```
x <- array(runif(4 * 3 * 10), c(4, 3, 10))
plotPressureMap(pressureMovie(x, 100, dx = 5, dy = 5))
```

---

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

```
sk <- define_skeleton("BODY_25")
labels <- sk$keypoints$label
set.seed(42)
nk <- length(labels); nf <- 10
mk <- function(b) matrix(rep(b, each = nf), nf)
px <- mk(rnorm(nk, 0, 0.3)); py <- mk(rnorm(nk, 0, 0.3))
pz <- mk(seq(1.8, 0, length.out = nk))
colnames(px) <- colnames(py) <- colnames(pz) <- labels
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    position_x = px, position_y = py, position_z = pz),
  colData = S4Vectors::DataFrame(label = labels, type = rep("keypoint", nk)),
  samplingRate = 30)
plotSkeleton(pe, sk, frame = 1)
```

---

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

```
sk <- define_skeleton("BODY_25")
labels <- sk$keypoints$label
set.seed(42)
nk <- length(labels); nf <- 10
mk <- function(b) matrix(rep(b, each = nf), nf)
px <- mk(rnorm(nk, 0, 0.3)); py <- mk(rnorm(nk, 0, 0.3))
pz <- mk(seq(1.8, 0, length.out = nk))
colnames(px) <- colnames(py) <- colnames(pz) <- labels
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    position_x = px, position_y = py, position_z = pz),
  colData = S4Vectors::DataFrame(label = labels, type = rep("keypoint", nk)),
  samplingRate = 30)
plotSkeleton3D(pe, sk, frame = 1)
```

---

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

```
sk <- define_skeleton("BODY_25")
labels <- sk$keypoints$label
set.seed(42)
nk <- length(labels); nf <- 10
mk <- function(b) matrix(rep(b, each = nf), nf)
px <- mk(rnorm(nk, 0, 0.3)); py <- mk(rnorm(nk, 0, 0.3))
pz <- mk(seq(1.8, 0, length.out = nk))
colnames(px) <- colnames(py) <- colnames(pz) <- labels
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    position_x = px, position_y = py, position_z = pz),
  colData = S4Vectors::DataFrame(label = labels, type = rep("keypoint", nk)),
  samplingRate = 30)
plotSkeletonOverlay(pe, sk, frames = c(1, 5, 10))
```

---

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

```
sk <- define_skeleton("BODY_25")
labels <- sk$keypoints$label
set.seed(42)
nk <- length(labels); nf <- 10
mk <- function(b) matrix(rep(b, each = nf), nf)
px <- mk(rnorm(nk, 0, 0.3)); py <- mk(rnorm(nk, 0, 0.3))
pz <- mk(seq(1.8, 0, length.out = nk))
colnames(px) <- colnames(py) <- colnames(pz) <- labels
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    position_x = px, position_y = py, position_z = pz),
  colData = S4Vectors::DataFrame(label = labels, type = rep("keypoint", nk)),
  samplingRate = 30)
plotSkeletonSequence(pe, sk, frames = c(1, 5, 10))
```

---

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.

Examples

```
set.seed(1)
waveforms <- matrix(sin(seq(0, pi, length.out = 101)) + rnorm(101 * 8, 0, 0.1),
                    101, 8)
plotSpaghetti(waveforms)
```

---

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.

Examples

```
set.seed(1)
plotTrajectory(cumsum(rnorm(100)), cumsum(rnorm(100)))
```

---

plotTUG *Plot an instrumented TUG timeline*

Description

Plots the trunk yaw angular velocity with the detected turns shaded and the sit-to-stand / turn-to-sit transitions marked.

Usage

```
plotTUG(report, title = "Instrumented TUG")
```

Arguments

report	An itug_report from instrumentedTUG().
title	Plot title.

Value

A ggplot object.

See Also

instrumentedTUG()

Examples

```
fs <- 100
t <- (0:(13 * fs - 1)) / fs
g <- function(t, mu, s) exp(-((t - mu) / s)^2)
yaw <- (pi / (0.6 * sqrt(pi))) * g(t, 5, 0.6) +
       (pi / (0.5 * sqrt(pi))) * g(t, 9.75, 0.5)
pitch <- 1.5 * g(t, 1, 0.3) + 1.3 * g(t, 11.5, 0.3)
av <- cbind(0.05 * sin(2 * pi * t), pitch, yaw)
plotTUG(instrumentedTUG(av, fs))
```

---

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()

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  set.seed(1)
  d <- matrix(rnorm(1000), 100, 10)
  plotUMAP(waveformUMAP(d, n_neighbors = 5))
}
```

---

plotWaveformComparison

Plot waveform comparison across groups

Description

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

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.

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)")
```

---

poseFix *Detect and correct pose-estimation anomalies*

Description

Cleans a markerless-pose recording (e.g. from readOpenPose()) by detecting biomechanically implausible keypoints and correcting them, connecting pose estimation to the downstream joint-angle / gait / kinematics analysis. Anomalies are flagged from four criteria (after Sugiyama, Uno & Matsui 2023): low detector confidence; segment (bone) lengths that deviate from the subject's own median; frame-to-frame jumps larger than a fraction of body scale; and joint angles outside their empirical range. Flagged coordinates are set to NA, linearly interpolated, and optionally smoothed. References are derived from the recording itself, so the method is model-agnostic.

Usage

```
poseFix(
  pe,
  conf_threshold = 0.2,
  length_tol = 0.4,
  jump_tol = 0.5,
  angle_range = c(0.01, 0.99),
  skeleton = NULL,
  joints = NULL,
  deswap = TRUE,
  length_correct = FALSE,
  smooth = TRUE,
  smooth_spar = 0.4
)
```

Arguments

pe	A PhysioExperiment of pose keypoints with assays keypoint_x, keypoint_y and (optionally) confidence, and colData$label naming the keypoints — the output of readOpenPose() / readMediaPipe().
conf_threshold	Keypoints with confidence below this are flagged (default 0.2).
length_tol	A bone is flagged when its length deviates from its own median by more than this fraction (default 0.4).
jump_tol	A keypoint is flagged when its frame-to-frame displacement exceeds this fraction of body scale (default 0.5).
angle_range	Lower/upper quantiles bounding plausible joint angles (default c(0.01, 0.99)).
skeleton	Optional list of c(from, to) keypoint-name bone pairs (default: a standard body skeleton restricted to the present keypoints).
joints	Optional list of c(a, b, c) angle triplets (angle at b).
deswap	If TRUE (default), first correct left/right leg-label swaps by restoring trajectory continuity (a common pose-estimation error during gait).
length_correct	If TRUE, additionally standardise segment lengths with poseLengthCorrect() after cleaning (camera-distance correction; default FALSE).
smooth	If TRUE (default) smooth each coordinate with a spline after interpolation.
smooth_spar	Smoothing parameter passed to stats::smooth.spline().

Value

A cleaned PhysioExperiment (corrected keypoint_x/keypoint_y assays); metadata()$poseFix holds the per-criterion anomaly counts, the number of leg-swap corrections (leg_swaps), the flagged frame-by-keypoint logical matrix, and the flagged fraction.

References

Sugiyama S, Uno K, Matsui Y (2023). Types of anomalies in two-dimensional video-based gait analysis in uncontrolled environments. *PLOS Computational Biology* 19:e1009989. doi:10.1371/journal.pcbi.1009989

See Also

readOpenPose(), calculateJointAngles()

Examples

```
lab <- c("nose", "neck", "Rshoulder", "Relbow", "Rwrist", "Lshoulder",
         "Lelbow", "Lwrist", "Rhip", "Rknee", "Rankle", "Lhip", "Lknee", "Lankle")
set.seed(1); nf <- 60; nk <- length(lab)
mk <- function(mx) { m <- matrix(runif(nf * nk, 0, mx), nf); colnames(m) <- lab; m }
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    keypoint_x = mk(1920), keypoint_y = mk(1080), confidence = mk(1)),
  colData = S4Vectors::DataFrame(label = lab, type = rep("keypoint", nk)),
  samplingRate = 30)
pe_fixed <- poseFix(pe)
SummarizedExperiment::assayNames(pe_fixed)
```

---

poseLengthCorrect *Standardise segment lengths (camera-distance correction)*

Description

In 2D video, apparent bone lengths shrink and grow as the subject moves toward or away from the camera, biasing kinematics. This rescales each limb segment so its length matches a stable reference (the recording's median or mean of that segment), repositioning the distal keypoint along the bone direction and chaining proximal to distal. Generalised form of the length correction in PoseFixeR (Sugiyama, Uno & Matsui 2023).

Usage

```
poseLengthCorrect(pe, chains = NULL, reference = c("median", "mean"))
```

Arguments

pe	A pose PhysioExperiment (assays keypoint_x, keypoint_y; colData$label), e.g. from readOpenPose() or after poseFix().
chains	Optional list of kinematic chains, each a proximal-to-distal vector of keypoint names (default: both legs and both arms, restricted to the present keypoints).
reference	"median" (default, robust) or "mean" reference length.

Value

A PhysioExperiment with length-standardised keypoint_x/y; metadata()$poseLengthCorrect holds the reference length per segment.

References

Sugiyama S, Uno K, Matsui Y (2023). *PLOS Comput Biol* 19:e1009989. doi:10.1371/journal.pcbi.1009989

See Also

poseFix()

Examples

```
lab <- c("nose", "neck", "Rshoulder", "Relbow", "Rwrist", "Lshoulder",
         "Lelbow", "Lwrist", "Rhip", "Rknee", "Rankle", "Lhip", "Lknee", "Lankle")
set.seed(1); nf <- 60; nk <- length(lab)
mk <- function(mx) { m <- matrix(runif(nf * nk, 0, mx), nf); colnames(m) <- lab; m }
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    keypoint_x = mk(1920), keypoint_y = mk(1080), confidence = mk(1)),
  colData = S4Vectors::DataFrame(label = lab, type = rep("keypoint", nk)),
  samplingRate = 30)
pe_corr <- poseLengthCorrect(pe)
SummarizedExperiment::assayNames(pe_corr)
```

---

predictiveSimulation *Predictive simulation: minimal-effort optimal control*

Description

Predicts the movement a limb model would produce to go from one posture to another while minimising control effort (integral of squared joint torque) – a direct-collocation optimal-control problem. Each joint trajectory is a boundary-satisfying quintic plus interior basis functions whose amplitudes are optimised; the required torque at each node is obtained by inverse dynamics.

Usage

```
predictiveSimulation(
  model,
  q0,
  qT,
  duration,
  n_nodes = 60L,
  n_basis = 3L,
  maxit = 200L
)
```

Arguments

model	A planar_limb_model.
q0, qT	Start and target joint configurations (length ndof); velocity is zero at both ends.
duration	Movement time (s).
n_nodes	Collocation nodes for the effort integral (default 60).
n_basis	Interior basis functions per joint (default 3).
maxit	Optimiser iterations (default 200).

Value

a `predictive_sim`: `time`, `q`, `qd`, `qdd`, `tau` (node x ndof), `effort` (optimised) and `effort_baseline` (the minimum-jerk quintic), the effort reduction, and boundary error.

References

Todorov & Jordan (2002); Ackermann & van den Bogert (2010) predictive simulation.

See Also

`forwardDynamics()`

Examples

```
m <- planarLimbModel(c(2.1, 1.4), c(0.3, 0.25), g = 0)   # horizontal reach
ps <- predictiveSimulation(m, q0 = c(0.2, 0.3), qT = c(1.1, 0.6),
                           duration = 0.6, n_basis = 2, maxit = 80)
ps$effort <= ps$effort_baseline                          # optimisation helps
```

`pressureAsymmetry` *Compare left and right plantar pressure*

Description

Calculates the signed Robinson symmetry index:

$$100(L - R)/((L + R)/2).$$

Usage

```
pressureAsymmetry(
  left,
  right,
  metric = c("peak_pressure", "pti", "contact_area", "total_force"),
  force_factor = 0.001
)
```

Arguments

`left`, `right`	Left and right `pressure_movie` objects.
`metric`	Scalar metric to compare.
`force_factor`	Conversion from pressure times cell area to force.

Value

A `pressure_asymmetry` object containing side values, signed and absolute symmetry indices, and the left/right ratio.

References

Robinson RO, Herzog W, Nigg BM (1987). Use of force platform variables to quantify the effects of chiropractic manipulation on gait symmetry. *Journal of Manipulative and Physiological Therapeutics*, 10:172-176.

Examples

```
left <- pressureMovie(array(100, c(3, 3, 5)), 100, side = "left")
right <- pressureMovie(array(80, c(3, 3, 5)), 100, side = "right")
pressureAsymmetry(left, right)
```

---

pressureMovie *Construct a plantar-pressure movie*

Description

Creates a compact representation of a pressure-sensor grid sampled over time. Pressure values are expected in kPa, and grid pitch is expressed in mm. Negative sensor values are clamped to zero.

Usage

```
pressureMovie(
  pressure,
  sampling_rate,
  dx = 1,
  dy = 1,
  side = NA_character_,
  units = "kPa",
  heel_first = TRUE
)
```

Arguments

pressure	Numeric three-dimensional array with dimensions grid row by grid column by frame, or a list of equal-dimension numeric matrices.
sampling_rate	Sampling frequency in Hz.
dx, dy	Sensor-cell pitch in the mediolateral and anteroposterior directions, respectively, in mm.
side	Optional foot side, "left", "right", or NA.
units	Pressure units recorded in the source data. Calculations assume kPa.
heel_first	Logical; whether grid row 1 is the posterior (heel) end.

Value

A pressure_movie object.

References

Pataky TC (2012). Spatial resolution in plantar pressure measurement revisited. *Journal of Biomechanics*, 45:2116-2124.

Examples

```
x <- array(0, dim = c(4, 3, 10))
x[2, 2, ] <- 100
pressureMovie(x, sampling_rate = 100, dx = 5, dy = 5)
```

---

pressureTimeIntegral *Calculate the plantar pressure-time integral*

Description

Uses trapezoidal integration over the sampled duration. The force-time integral converts kPa times mm-squared to N using `force_factor`.

Usage

```
pressureTimeIntegral(pm, force_factor = 0.001)
```

Arguments

`pm`	A `pressure_movie`.
`force_factor`	Conversion from pressure times cell area to force. The default `1e-3` converts kPa times mm-squared to N; use `0.1` for kPa times cm-squared.

Value

A numeric matrix of pressure-time integrals in kPa-s. Attribute `"fti"` is the total force-time integral in N-s.

Examples

```
pm <- pressureMovie(array(100, c(2, 2, 11)), 100, dx = 5, dy = 5)
pressureTimeIntegral(pm)
```

`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.

Examples

```
f <- system.file("testdata", "sample.asf", package = "PhysioMoCap")
if (nzchar(f)) {
  skeleton <- readASF(f)
  print(skeleton)
}
```

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.

Examples

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

`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.

Examples

```
ex <- createBenchmarkExample(n_trials = 2, seed = 1)
print(validateBenchmarkManifest(ex$manifest))
```

`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.

Examples

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

Examples

```
set.seed(123)
vGRF <- c(rep(0, 100), sin(seq(0, pi, length.out = 500)) * 800, rep(0, 400))
vGRF <- vGRF + rnorm(1000, 0, 10)
ev <- detectEvents(as.matrix(vGRF), schema_gait,
  signals = list(vGRF = vGRF), sampling_rate = 1000)
print(ev)
```

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)

Examples

```
set.seed(123)
tt <- seq(0, 2 * pi, length.out = 50)
g1 <- sapply(1:5, function(i) sin(tt) + rnorm(50, 0, 0.1))
g2 <- sapply(1:5, function(i) cos(tt) + rnorm(50, 0, 0.1))
print(dtwClustering(cbind(g1, g2), k = 2))
```

---

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)

Examples

```
x <- sin(seq(0, pi, length.out = 100)) * 30
y <- sin(seq(0, pi, length.out = 100) + 0.3) * 30
print(dtwDistance(x, y))
```

---

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.

Examples

```
n <- 500
z <- matrix(0, n, 1, dimnames = list(NULL, "fp1"))
fz <- matrix(c(rep(0, 100), rep(700, 300), rep(0, 100)), n, 1,
             dimnames = list(NULL, "fp1"))
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    force_x = z, force_y = z, force_z = fz,
    moment_x = matrix(50, n, 1, dimnames = list(NULL, "fp1")),
    moment_y = matrix(-100, n, 1, dimnames = list(NULL, "fp1")),
    moment_z = z),
  colData = S4Vectors::DataFrame(label = "fp1", type = "forceplate"),
  samplingRate = 1000)
print(analyzeForcePlatePE(pe, threshold = 20, cutoff = 20,
  filter_method = "moving_average"))
```

---

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.

Examples

```
n <- 500
lab <- c("fp1", "fp2")
z <- matrix(0, n, 2, dimnames = list(NULL, lab))
fz <- matrix(c(rep(0, 100), rep(700, 300), rep(0, 100)), n, 2,
             dimnames = list(NULL, lab))
pe <- PhysioExperiment(assays = S4Vectors::SimpleList(
    force_x = z, force_y = z, force_z = fz,
    moment_x = matrix(50, n, 2, dimnames = list(NULL, lab)),
    moment_y = matrix(-100, n, 2, dimnames = list(NULL, lab)),
    moment_z = z),
  colData = S4Vectors::DataFrame(label = lab, type = c("forceplate", "forceplate")),
  samplingRate = 1000)
print(analyzeForcePlatePE(pe, plate_index = "all", threshold = 20,
  cutoff = 20, filter_method = "moving_average"))
```

`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.

Examples

```
demo <- demoMoCapData(seed = 1)
ev <- detectEventsZeni(demo$mocap,
  markers = list(heel_right = "Ankle_R", toe_right = "Toe_R",
                 heel_left = "Ankle_L", toe_left = "Toe_L"),
  reference = "Pelvis_R")
print(calculateGaitParameters(demo$mocap, ev))
```

---

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.

Examples

```
qs <- quickStartMoCap(seed = 1)
print(qs)
```

`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.

Examples

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

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
phases <- segmentPhases(data, events, schema_gait)
print(combineTrials(phases, phases, labels = c("t1", "t2")))
```

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)

Examples

```
set.seed(123)
data <- matrix(rnorm(200), 100, 2)
events <- data.frame(
  event = c("hs1","ff","ms","ho","to","hs2"),
  label = c("Heel Strike","Foot Flat","Midstance","Heel Off","Toe Off","Heel Strike 2"),
  index = c(1, 13, 31, 41, 61, 100), time = c(0, .12, .30, .40, .60, 1.0),
  percent = c(0, 12, 30, 40, 60, 100), method = rep("manual", 6),
  confidence = rep(1, 6), stringsAsFactors = FALSE)
class(events) <- c("detected_events", "data.frame")
attr(events, "sampling_rate") <- 100
print(segmentPhases(data, events, schema_gait))
```

---

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.

Examples

```
kp <- data.frame(id = 0:2, label = c("Head", "Torso", "Hip"),
                 body_region = c("head", "torso", "pelvis"),
                 stringsAsFactors = FALSE)
bones <- data.frame(from_id = c(0, 1), to_id = c(1, 2),
                    bone_name = c("neck", "spine"), stringsAsFactors = FALSE)
sk <- SkeletonModel("mini", kp, bones, list(Head = "Torso", Torso = "Hip"), "Head")
print(sk)
```

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)

Examples

```
set.seed(1)
d <- matrix(rnorm(1000), 100, 10)
print(waveformPCA(d, method = "features"))
```

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)

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  set.seed(1)
  d <- matrix(rnorm(1000), 100, 10)
  print(waveformUMAP(d, n_neighbors = 5))
}
```

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)
```

promCondition *Condition a ProMP on a via-point*

Description

Gaussian conditioning of a fitted promFit() ProMP so the trajectory distribution passes through a desired point at a given phase, updating the mean and the variability band.

Usage

```
promCondition(promp, phase, value, obs_sd = 0.001)
```

Arguments

promp	A promp from promFit().
phase	Phase in [0, 1] of the via-point.
value	Desired trajectory value at phase.
obs_sd	Observation noise SD (smaller = tighter conditioning; default 1e-3).

Value

an updated promp with the conditioned mean, sd, w_mean, w_cov.

See Also

promFit()

Examples

```
z <- seq(0, 1, length.out = 50)
demos <- lapply(1:20, function(i) sin(2 * pi * z) + rnorm(50, 0, 0.1))
p <- promFit(demos)
cond <- promCondition(p, phase = 0.5, value = 1.0)
length(cond$mean)
```

promFit *Fit a Probabilistic Movement Primitive to demonstrations*

Description

Learns a distribution over trajectories from several demonstrations: each is projected onto a Gaussian basis, and the basis weights are modelled as Gaussian, giving a mean trajectory and a variability band.

Usage

```
promFit(demos, n_basis = 15L, ridge = 1e-06)
```

Arguments

demos	A list of demonstrations (each a numeric vector, all resampled to the same length), or an N x T matrix (one demo per row).
n_basis	Number of Gaussian basis functions (default 15).
ridge	Ridge penalty for the per-demo weight fit (default 1e-6).

Value

a promp object: mean and sd trajectories, the weight mean w_mean and covariance w_cov, the basis Phi, and phase z.

References

Paraschos A, et al. (2013) NIPS 26.

See Also

promCondition(), dmpFit()

Examples

```
z <- seq(0, 1, length.out = 50)
demos <- lapply(1:20, function(i) sin(2 * pi * z) + rnorm(50, 0, 0.1))
p <- promFit(demos)
length(p$mean)                                    # mean trajectory
```

---

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
```

---

rateOfForceDevelopment

Rate of force development

Description

Computes sequential interval RFD from contraction onset and the maximum moving-window slope. RFD is expressed in N m/s.

Usage

```
rateOfForceDevelopment(
  torque,
  sampling_rate,
  onset = NULL,
  onset_method = c("absolute", "sd"),
  onset_threshold = 7.5,
  baseline_ms = 100,
  baseline_sd_k = 5,
  windows_ms = c(50, 100, 200),
  peak_window_ms = 20
)
```

Arguments

torque	Numeric torque trace in N m.
sampling_rate	Sampling rate in Hz.
onset	Optional 1-based integer onset sample. When NULL, onset is detected from the baseline.
onset_method	Either a fixed "absolute" increase above baseline or a baseline "sd" rule.
onset_threshold	Absolute onset threshold in N m above baseline.
baseline_ms	Baseline duration in milliseconds.
baseline_sd_k	Number of baseline SDs used by the "sd" method.
windows_ms	Positive interval widths in milliseconds.
peak_window_ms	Positive moving-window width in milliseconds.

Value

An rfd object containing onset details, interval RFD values, and peak moving-window RFD.

References

Aagaard P, Simonsen EB, Andersen JL, Magnusson P, Dyhre-Poulsen P (2002). Increased rate of force development and neural drive of human skeletal muscle following resistance training. *Journal of Applied Physiology*, 93:1318-1326. doi:10.1152/japplphysiol.00283.2002

Maffiuletti NA, Aagaard P, Blazevich AJ, Folland J, Tillin N, Duchateau J (2016). Rate of force development: physiological and methodological considerations. *European Journal of Applied Physiology*, 116:1091-1116. doi:10.1007/s0042101633466

Examples

```
torque <- c(rep(0, 100), seq(0.4, 120, by = 0.4), rep(120, 100))
rateOfForceDevelopment(torque, sampling_rate = 1000, onset_method = "sd")
```

---

reachingKinematics *Reaching kinematics report for a PhysioExperiment*

Description

Extracts a hand-marker trajectory, computes tangential speed with computeVelocity() and computeSpeed(), and reports temporal, submovement, smoothness, endpoint, and optional trunk-compensation metrics.

Usage

```
reachingKinematics(
  pe,
  marker,
  target = NULL,
  assay_prefix = "position",
  sampling_rate = NULL,
  onset_threshold = 0.05,
  trunk_marker = NULL,
  shoulder_markers = NULL,
  ...
)
```

Arguments

pe	A PhysioExperiment with position_x, position_y, and optionally position_z assays.
marker	Reaching hand marker name or column index.
target	Optional target coordinate matching trajectory dimension.
assay_prefix	Position-assay prefix (default "position").
sampling_rate	Optional sampling frequency in Hz; defaults to PhysioExperiment::samplingRate().
onset_threshold	Movement threshold.
trunk_marker	Optional trunk marker name or column index.
shoulder_markers	Optional length-2 vector identifying right and left shoulder markers.
...	Additional arguments passed to sparc().

Value

A reaching_kinematics report with movement time, peak velocity, time to peak, movement units, SPARC, LDLJ, dimensionless jerk, movement bounds, sampling rate, marker, and optional endpoint/trunk results.

Examples

```
demo <- demoMoCapData(seed = 1)
rk <- reachingKinematics(demo$mocap, marker = "Toe_R")
names(rk)
```

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. Residual quality data are not exposed by this reader. 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

c3dEventTable(), 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

```
f <- system.file("testdata", "sample_dlc.csv", package = "PhysioMoCap")
if (nzchar(f)) {
  pe <- readDeepLabCut(f, format = "csv")
  pe
}
```

---

readGaitRec *Read GaitRec Dataset*

Description

Reads ground reaction force (GRF) data or spatiotemporal parameters from the GaitRec dataset format (Horsak et al., 2020). GaitRec is a large-scale clinical gait dataset containing GRF waveforms and gait parameters from patients with various lower-limb orthopaedic conditions and healthy controls.

Usage

```
readGaitRec(
  path,
  type = c("grf", "parameters"),
  sr = 1000,
  sep = ",",
  subject_id = NULL
)
```

Arguments

`path`	Path to a GaitRec CSV file or a directory containing GaitRec CSV files. When a directory is given, all CSV files matching the specified `type` are read and combined.
`type`	Type of data to read: `"grf"` for ground reaction force waveforms or `"parameters"` for spatiotemporal parameters. Default `"grf"`.
`sr`	Sampling rate in Hz. Default `1000` for GaitRec GRF data. For time-normalised waveforms (101 data points representing 0–100\ gait cycle), this value is used as a nominal sampling rate in the resulting PhysioExperiment object.

sep	Column separator in the CSV file. Default ",". Set to "\t" for tab-separated files.
subject_id	Optional subject identifier string to store in the returned object's metadata. When path is a directory this is ignored and identifiers are extracted from file names.

Details

GaitRec GRF file format:

GaitRec GRF CSV files typically contain columns for time and force components from two force plates:

- A time column (e.g., "time", "Time", "t")
- Force columns following naming patterns like "Fx1", "Fy1", "Fz1", "Fx2", "Fy2", "Fz2" (plate number suffix) or "FP1_Fx", "FP1_Fy", "FP1_Fz", "FP2_Fx", etc. (plate prefix)

The function auto-detects force column naming patterns and extracts the component (x/y/z) and plate number. Columns that do not match recognised force patterns are stored in metadata(pe)$extra_columns as a data.frame.

Time-normalised waveforms (101 data points, 0–100\ also supported. The parser detects these by row count and stores a percent_gait_cycle vector in the metadata.

Value

For type = "grf": a PhysioExperiment object with a "raw" assay containing the GRF data matrix (time points x channels). The colData records channel labels, force component ("Fx", "Fy", "Fz"), plate number, and unit ("N"). Metadata includes source_file, format, and subject_id.

For type = "parameters": a data.frame with spatiotemporal gait parameters extracted from the CSV file.

References

Horsak B, Slijepcevic D, Raberger A-M, Schwab C, Worisch M, Zeppelzauer M (2020). "GaitRec, a large-scale ground reaction force dataset of healthy and impaired gait." *Scientific Data*, 7, 143. doi:10.1038/s415970200481z

See Also

readMoCapCSV() for generic CSV motion capture data, filterGRF() for low-pass filtering GRF signals, analyzeForcePlate() for force plate analysis.

Examples

```
# Write a small GaitRec-style GRF file, then read it back
tmp <- tempfile(fileext = ".csv")
header <- "time,GRF_X_1,GRF_Y_1,GRF_Z_1"
rows <- vapply(1:50, function(i) {
  paste(c(sprintf("%.3f", (i - 1) / 1000),
          sprintf("%.1f", c(10 + i, 500 + i * 2, 50 + i))), collapse = ",")
}, character(1))
writeLines(c(header, rows), tmp)
pe <- readGaitRec(tmp)
```

pe

---

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

```
set.seed(42)
cols <- list()
for (j in 0:32) {
  cols[[paste0("landmark_", j, "_x")]] <- runif(5)
  cols[[paste0("landmark_", j, "_y")]] <- runif(5)
  cols[[paste0("landmark_", j, "_z")]] <- runif(5, -0.1, 0.1)
  cols[[paste0("landmark_", j, "_visibility")]] <- runif(5, 0.5, 1.0)
}
path <- tempfile(fileext = ".csv")
write.csv(as.data.frame(cols), path, row.names = FALSE)
pe <- readMediaPipe(path, model = "pose", fps = 30)
pe
```

---

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

```
path <- tempfile(fileext = ".csv")
df <- data.frame(Marker1_X = rnorm(10), Marker1_Y = rnorm(10),
                 Marker1_Z = rnorm(10))
write.csv(df, path, row.names = FALSE)
pe <- readMoCapCSV(path, format = "wide", sampling_rate = 100)
pe
```

---

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()`

Requires the OpenCap web API (network access and an API key), so the example cannot run offline and is not executed.

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

```
set.seed(123)
tmp <- tempfile(fileext = ".json")
people <- list(list(person_id = list(-1L),
                    pose_keypoints_2d = as.list(runif(25 * 3, 0, 500))))
jsonlite::write_json(list(version = 1.3, people = people),
                     tmp, auto_unbox = TRUE)
pe <- readOpenPose(tmp, model = "BODY_25", fps = 30)
pe
```

---

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()

Examples

```
f <- system.file("testdata", "sample.mot", package = "PhysioMoCap")
if (nzchar(f)) {
  out <- readOpenSimOutputs(f)
  out
}
```

`readPressureFrame` *Read a plantar-pressure ASCII export*

Description

Reads frame-delimited Novel/Tekscan-style text exports. Comment metadata may define `sampling_rate` (or `fs`), `dx`, `dy`, `side`, and `units`. Explicit function arguments take precedence over file metadata.

Usage

```
readPressureFrame(
  file,
  sampling_rate = NULL,
  dx = NULL,
  dy = NULL,
  side = NA_character_,
  units = "kPa",
  heel_first = TRUE,
  frame_marker = "^\\s*(Frame|ASCII_DATA|Time)",
  comment = "#"
)
```

Arguments

`file`	Path to an ASCII pressure export.
`sampling_rate, dx, dy`	Optional overrides for file metadata.
`side`	Optional side override.
`units`	Pressure-unit override. When omitted, file metadata is used, falling back to `"kPa"`.
`heel_first`	Logical; whether row 1 represents the heel.
`frame_marker`	Regular expression identifying frame-header lines.
`comment`	Literal prefix identifying metadata/comment lines.

Value

A `pressure_movie` object.

Examples

```
f <- tempfile(fileext = ".txt")
writeLines(c("# sampling_rate 100", "Frame 0", "1 2", "3 4"), f)
readPressureFrame(f)
unlink(f)
```

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

```
f <- system.file("testdata", "sample_venus3d.csv", package = "PhysioMoCap")
if (nzchar(f)) {
  pe <- readVenus3D(f)
  pe
}
```

`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")
```

`recurrenceQuantification` *Recurrence quantification analysis (RQA)*

Description

Reconstructs the attractor by delay embedding and quantifies its recurrence structure: recurrence rate, determinism, laminarity and diagonal/vertical line statistics. High determinism indicates deterministic (periodic/chaotic) rather than stochastic dynamics.

Usage

```
recurrenceQuantification(
  x,
  m = 3L,
  tau = 1L,
  radius = NULL,
  target_rr = 0.1,
  lmin = 2L,
  vmin = 2L,
  max_n = 1500L
)
```

Arguments

`x`	Numeric time series.
`m`	Embedding dimension (default 3).
`tau`	Embedding delay (default 1).
`radius`	Recurrence threshold; if `NULL`, chosen to reach `target_rr`.
`target_rr`	Target recurrence rate when `radius` is `NULL` (default 0.1).
`lmin`, `vmin`	Minimum diagonal / vertical line lengths (default 2).
`max_n`	Cap on embedded points (subsampled if longer; default 1500).

Value

an `rqa_result`: `RR`, `DET`, `LAM`, `Lmax`, `Lmean`, `TT` (trapping time), `ENTR` (diagonal line entropy), `radius`.

References

Marwan N, et al. (2007) Phys Rep 438:237-329.

See Also

`floquetMultipliers()`, `approximateEntropy()`

Examples

```
t <- seq(0, 20 * pi, length.out = 800)
recurrenceQuantification(sin(t))$DET              # ~1 (deterministic)
```

`reflexThreshold` *Velocity-dependent reflex threshold*

Description

Regresses the catch angle against the peak stretch velocity across a series of passive stretches at varying velocities. A negative slope is the signature of velocity-dependent spasticity (the catch occurs earlier - at a smaller angle - as the stretch gets faster).

Usage

```
reflexThreshold(stretches)
```

Arguments

`stretches`	A list of `tardieuStretch()` results (>= 2), one per stretch.

Value

An S3 `reflex_threshold` list: `slope` (deg per deg/s), `intercept`, `catch_angles`, `peak_velocities` and the linear `model`.

See Also

`tardieuStretch()`, `tardieuScore()`

Examples

```
fs <- 200
mk <- function(catch, rise, total = 0.8) {
  t <- seq(0, total, by = 1 / fs)
  ifelse(t <= rise, catch * t / rise, catch + 8 * (t - rise) / (total - rise))
}
st <- lapply(list(c(30, .5), c(20, .25), c(15, .13)),
             function(p) tardieuStretch(mk(p[1], p[2]), sampling_rate = fs,
                                        onset = "velocity"))
reflexThreshold(st)$slope
```

`regionalLoading` *Summarize plantar loading by anatomical region*

Description

Partitions the active footprint along its anteroposterior extent using cumulative Cavanagh-style region bounds.

Usage

```
regionalLoading(
  pm,
  statistic = c("fti", "peak_force", "mean_force"),
  regions = c(rearfoot = 0.3, midfoot = 0.6, forefoot = 0.8, toes = 1),
  contact_threshold = 0,
  force_factor = 0.001
)
```

Arguments

`pm`	A `pressure_movie`.
`statistic`	Regional statistic: force-time integral, peak force, or mean force.
`regions`	Named, strictly increasing cumulative upper bounds ending at 1. Defaults to rearfoot, midfoot, forefoot, and toes.
`contact_threshold`	Pressure threshold in kPa defining loaded cells.
`force_factor`	Conversion from pressure times cell area to force.

Value

A `regional_loading` data frame with one row per region. Attributes contain the regional total and selected statistic.

References

Cavanagh PR, Rodgers MM, Iiboshi A (1987). Pressure distribution under symptom-free feet during barefoot standing. *Foot and Ankle*, 7:262-276.

Examples

```
pm <- pressureMovie(array(100, c(10, 4, 11)), 100, dx = 5, dy = 5)
regionalLoading(pm)
```

`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 <- PhysioExperiment::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)
PhysioExperiment::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 when any tool returns non-zero status.
`extra_args`	Optional character vector appended to every tool run.

Details

Requires a native OpenSim installation and tool setup XML files, so the example cannot run offline and is not executed.

Value

Named list with entries for tools that were executed (`ik`, `id`, `so`, `rra`, `cmc`, `analyze`).

See Also

batch_analyze_opensim(), create_schema_from_opensim()

Examples

```
## Not run:
# Requires a native OpenSim installation and OpenSim setup XML files.
run_opensim_toolchain(ik_setup = "ik_setup.xml")

## End(Not run)
```

---

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
```

---

runCMC *Run Computed Muscle Control (CMC) via OpenSim*

Description

Orchestrates the OpenSim Computed Muscle Control tool through PhysioOpenSim. CMC has no pure-R fallback, so it requires a working OpenSim backend.

Usage

```
runCMC(
  cmc_setup,
  workdir = NULL,
  cli = NULL,
  timeout_sec = 0L,
  fail_on_error = TRUE
)
```

Arguments

cmc_setup	Path to an OpenSim CMC setup XML.
workdir, cli, timeout_sec, fail_on_error	Passed to PhysioOpenSim::opensimRunCMC().

Details

Requires a native OpenSim installation and a CMC setup XML file, so the example cannot run offline and is not executed.

Value

An opensim_tool_result list with tool (the PhysioOpenSim result).

See Also

runStaticOptimization(), runRRA()

Examples

```
## Not run:
# Requires a native OpenSim installation and a CMC setup XML file.
runCMC("cmc_setup.xml")

## End(Not run)
```

`runOpenSimFromMarkers` *Run the local OpenSim toolchain from a model and marker file*

Description

Convenience wrapper that wires markers through OpenSim inverse kinematics (IK) and, optionally, inverse dynamics (ID) and static optimization (SO): it writes the tool setup XMLs from the bundled **PhysioOpenSim** templates, runs them in order via `run_opensim_toolchain()`, and parses the results with `readOpenSimOutputs()` into a downstream-ready form. This is the local counterpart to using OpenCap's cloud kinematics directly.

Usage

```
runOpenSimFromMarkers(
  model_file,
  trc_file,
  tools = "ik",
  external_loads_file = NULL,
  time_range = NULL,
  workdir = NULL,
  templates = NULL,
  cli = NULL,
  require_external_loads = TRUE,
  dry_run = FALSE
)
```

Arguments

`model_file`	Path to a scaled OpenSim `.osim` model (e.g. from `downloadOpenCapModel()`).
`trc_file`	Path to a marker trajectory `.trc` file.
`tools`	Which tools to run, a subset of `"ik"`, `"id"`, `"so"` (IK is always required and run first). Default `"ik"`.
`external_loads_file`	Path to an OpenSim `ExternalLoads` XML (ground reactions). Required for `"id"`/`"so"` unless `require_external_loads` is `FALSE`.
`time_range`	Optional numeric `c(start, end)` (s) applied to every tool. If `NULL` (default) the whole trial is used, read from `trc_file` (otherwise the tool templates' own `<time_range>` would truncate the trial to 0-1 s).
`workdir`	Directory for setup XMLs and outputs (a temporary directory by default).

templates — Optional named list of template paths (ik/id/so); defaults to the templates bundled with **PhysioOpenSim**.

cli — Optional path to the opensim-cmd CLI (see run_opensim_toolchain()).

require_external_loads — If TRUE (default) error when "id"/"so" are requested without external_loads_file.

dry_run — If TRUE, only write the setup XMLs and return them without a working OpenSim backend (useful for inspection/testing). Default FALSE.

Details

IK needs only the scaled model and markers. **ID and SO additionally need ground-reaction data** (external_loads_file); markerless OpenCap has no force plates, so you must supply a measured or estimated ExternalLoads XML (or set require_external_loads = FALSE to run kinematics-only inverse dynamics, which is valid only for non-contact phases such as swing). CMC and RRA need actuator/tracking-task files beyond what this wrapper infers; run them via run_opensim_toolchain() with the corresponding PhysioOpenSim::opensimWrite*SetupFromTemplate() writers.

Requires a native OpenSim installation (SystemRequirements: OpenSim) and setup/model files, so the example cannot run offline and is not executed.

Value

An opensim_run list: setups (written XML paths), workdir, expected_outputs; and, when run, toolchain (raw run_opensim_toolchain() result), files (produced .mot/.sto), outputs (their parsed PhysioExperiment objects from readOpenSimOutputs()), motion (the IK joint-angle object), and backend.

See Also

runOpenSimFromOpenCap(), run_opensim_toolchain(), readOpenSimOutputs(), downloadOpenCapModel()

Examples

```
## Not run:
res <- runOpenSimFromMarkers(
  model_file = "subject_scaled.osim", trc_file = "walk.trc",
  tools = c("ik", "id", "so"), external_loads_file = "walk_grf.xml")
res$motion            # IK joint angles (PhysioExperiment)
calculateJointAngles(res$motion)

## End(Not run)
```

runOpenSimFromOpenCap *Run the local OpenSim toolchain directly from an OpenCap session*

Description

End-to-end bridge: downloads an OpenCap session's scaled OpenSim model (downloadOpenCapModel()) and marker .trc, then runs the local OpenSim toolchain (runOpenSimFromMarkers()). Requires a working OpenSim backend (native **PhysioOpenSim** build or the opensim-cmd CLI) and, for ID/SO, an external_loads_file (OpenCap is markerless and has no ground reactions).

Usage

```
runOpenSimFromOpenCap(
  session_id,
  trial_id = NULL,
  api_key = NULL,
  base_url = "https://app.opencap.ai/api",
  model_file = NULL,
  tools = "ik",
  external_loads_file = NULL,
  time_range = NULL,
  workdir = NULL,
  templates = NULL,
  cli = NULL,
  require_external_loads = TRUE,
  dry_run = FALSE
)
```

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.
base_url	Character string giving the base URL for the OpenCap API. Default is "https://app.opencap.ai/api".
model_file	Optional local .osim model; if NULL (default) it is downloaded from the session with downloadOpenCapModel().
tools, external_loads_file, time_range, workdir, templates, cli, require_external_loads, dry_run	Passed to runOpenSimFromMarkers().

Details

With `dry_run = TRUE` the model and markers are still downloaded (they are needed to write the setups); only the OpenSim tool execution is skipped. For a fully offline dry run, call `runOpenSimFromMarkers()` with local files.

Requires the OpenCap web API (network + credentials) and a native OpenSim installation, so the example cannot run offline and is not executed.

Value

An `opensim_run` object (see `runOpenSimFromMarkers()`).

See Also

`runOpenSimFromMarkers()`, `readOpenCap()`, `downloadOpenCapModel()`

Examples

```
## Not run:
res <- runOpenSimFromOpenCap(
  session_id = "abcd1234-5678-90ab-cdef-1234567890ab",
  tools = c("ik", "id", "so"), external_loads_file = "walk_grf.xml")
res$outputs

## End(Not run)
```

---

runRRA *Run Residual Reduction Algorithm (RRA) via OpenSim*

Description

Orchestrates the OpenSim Residual Reduction Algorithm through PhysioOpenSim. RRA has no pure-R fallback, so it requires a working OpenSim backend.

Usage

```
runRRA(
  rra_setup,
  workdir = NULL,
  cli = NULL,
  timeout_sec = 0L,
  fail_on_error = TRUE
)
```

Arguments

`rra_setup`	Path to an OpenSim RRA setup XML.
`workdir, cli, timeout_sec, fail_on_error`	Passed to `PhysioOpenSim::opensimRunRRA()`.

Details

Requires a native OpenSim installation and an RRA setup XML file, so the example cannot run offline and is not executed.

Value

An `opensim_tool_result` list with `tool` (the PhysioOpenSim result).

See Also

`runStaticOptimization()`, `runCMC()`

Examples

```
## Not run:
# Requires a native OpenSim installation and an RRA setup XML file.
runRRA("rra_setup.xml")

## End(Not run)
```

---

runStaticOptimization *Run static optimization (OpenSim, or a pure-R fallback)*

Description

Resolves net joint moments into muscle activations. When an OpenSim backend is available and a Static Optimization setup file is supplied, it runs the OpenSim Static Optimization tool via PhysioOpenSim; otherwise it falls back to `staticOptimizationR()`, the pure-R quadratic muscle-effort solver, using the supplied moment arms, maximum forces and joint moments.

Usage

```
runStaticOptimization(
  so_setup = NULL,
  moment_arms = NULL,
  max_force = NULL,
  joint_moments = NULL,
  execution = c("auto", "opensim", "r"),
  workdir = NULL,
  cli = NULL,
  timeout_sec = 0L,
  fail_on_error = TRUE,
  ...
)
```

Arguments

so_setup	Path to an OpenSim Static Optimization setup XML (OpenSim backend).
moment_arms, max_force, joint_moments	Inputs for the pure-R fallback, passed to staticOptimizationR().
execution	Backend selection: "auto" (default; OpenSim when available and so_setup is given, else pure-R), "opensim", or "r".
workdir, cli, timeout_sec, fail_on_error	Passed to PhysioOpenSim::opensimRunSO() for the OpenSim backend.
...	Additional arguments passed to staticOptimizationR() for the pure-R backend (e.g. weights, activation_bounds).

Value

An opensim_so_result list with backend ("opensim" or "r") and either tool (the PhysioOpenSim tool result) or static_optimization (the staticOptimizationR() result).

See Also

staticOptimizationR(), runRRA(), runCMC()

Examples

```
if (requireNamespace("quadprog", quietly = TRUE)) {
  set.seed(1)
  n_muscle <- 6; n_dof <- 2; n_frames <- 15
  r <- matrix(runif(n_dof * n_muscle, -0.05, 0.05), n_dof, n_muscle)
  fmax <- runif(n_muscle, 200, 1200)
  a <- matrix(runif(n_frames * n_muscle, 0, 0.6), n_frames, n_muscle)
  cmat <- sweep(r, 2, fmax, "*")
  tau <- t(apply(a, 1, function(ai) as.numeric(cmat %*% ai)))
  res <- runStaticOptimization(moment_arms = r, max_force = fmax,
    joint_moments = tau, execution = "r")
  res$backend
}
```

---

sampleEntropy *Sample entropy (Richman & Moorman)*

Description

Sample entropy quantifies the irregularity of a time series as the negative natural log of the conditional probability that sequences similar for m points remain similar at the next point, within tolerance r (self-matches excluded). Larger values indicate greater irregularity/complexity; a perfectly regular signal tends to zero.

Usage

```
sampleEntropy(x, m = 2L, r = 0.2, normalize = TRUE)
```

Arguments

x	Numeric time series.
m	Embedding (template) length (default 2).
r	Similarity tolerance. If normalize = TRUE (default) it is a multiple of the series standard deviation (default 0.2).
normalize	If TRUE, r is scaled by sd(x).

Value

A single numeric sample-entropy value (Inf if no length-m+1 matches occur).

References

Richman JS, Moorman JR (2000). Am J Physiol 278(6):H2039-H2049.

See Also

maxLyapunovExponent()

Examples

```
sampleEntropy(sin(seq(0, 40 * pi, length.out = 800)))  # ~0 (regular)
```

---

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)
```

---

scaleBodyModel *Scale a body model from marker-measured segment lengths*

Description

Measures each lower-limb segment length from joint-centre trajectories (the mean over frames of the distance between the segment's endpoints) and scales the de Leva model to the subject's body mass with those measured lengths.

Usage

```
scaleBodyModel(joints, body_mass, model = c("deLeva_male", "deLeva_female"))
```

Arguments

joints	A matrix/data.frame with ankle, toe, knee, hip joint coordinate columns (_x/_y, plus _z for 3D), as for inverseDynamicsRNE().
body_mass	Subject body mass in kg.
model	"deLeva_male" or "deLeva_female".

Value

A segment inertia data.frame with the marker-measured length.

See Also

scaleSegmentModel(), inverseDynamicsRNE()

Examples

```
n <- 10
joints <- data.frame(
  ankle_x = rep(0, n), ankle_y = rep(0.08, n), toe_x = rep(0.15, n),
  toe_y = rep(0.03, n), knee_x = rep(0, n), knee_y = rep(0.48, n),
  hip_x = rep(0, n), hip_y = rep(0.90, n))
scaleBodyModel(joints, body_mass = 70)
```

---

scalePlanarModel *Scale a planar chain model from a static pose*

Description

Estimates the fixed segment lengths of a serial planar chain from the marker positions of its nodes (joint centres and the end point) in a static/reference pose.

Usage

```
scalePlanarModel(nodes)
```

Arguments

nodes	A (n_segments + 1) x 2 matrix of node positions (root, joint 1, ..., end point) in the static pose.

Value

a planar_ik_model: lengths (per segment) and n_segments.

See Also

inverseKinematicsMarkers(), forwardKinematics2D()

Examples

```
nodes <- rbind(c(0, 0), c(0.4, 0), c(0.8, 0))      # two 0.4 m segments
scalePlanarModel(nodes)$lengths
```

`scaleSegmentModel` *Scale a de Leva segment inertia model to a subject*

Description

Scales the de Leva (1996) body-segment inertial parameters to a subject's body mass and segment lengths (or stature). A thin, explicitly-named wrapper around `estimateSegmentInertia()`.

Usage

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

Arguments

`body_mass`	Subject body mass in kg.
`segment_lengths`	Optional named vector of `foot`, `shank`, `thigh` lengths (m). If omitted, `body_height` is used with typical ratios.
`body_height`	Subject stature in m (used when `segment_lengths` is omitted).
`model`	`"deLeva_male"` or `"deLeva_female"`.

Value

A segment inertia data.frame (see `estimateSegmentInertia()`).

References

de Leva P (1996). "Adjustments to Zatsiorsky-Seluyanov's segment inertia parameters." J Biomech, 29(9), 1223-1230.

See Also

`scaleBodyModel()`, `estimateSegmentInertia()`, `inverseDynamicsRNE()`

Examples

```
scaleSegmentModel(body_mass = 68, body_height = 1.70)
```

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)
```

segmentCohortCycles *Segment cohort recordings while preserving subject and session identity*

Description

Reuses the PhysioCore cohort hierarchy and segmentMultimodalCycles(). Sessions must be MultiPhysioExperiment objects with at least two consistently named streams. No resampling or synchronization is performed.

Usage

```
segmentCohortCycles(
  x,
  cycles,
  assay = "raw",
  incomplete = c("error", "retain")
)
```

Arguments

x	A PhysioCore PhysioCohort.
cycles	Nonempty data frame with character subject_id, session_id, trial_id, side, cycle_id, and numeric start_time, end_time. Times are seconds relative to each session's master-clock t0, as in PhysioExperiment::streamTimeIndex(). Trial IDs describe intervals within a session; they are never inferred from session IDs. Additional columns are retained. An optional participant_id must equal subject_id.
assay	A single assay name used in every stream (default "raw"). Assays must be numeric time-by-channel matrices.
incomplete	Passed to segmentMultimodalCycles().

Details

Subject-level splits can be made using existing cohort subsetting before extraction, together with the corresponding cycle rows. This bridge does not enforce statistical independence or downstream training splits. Units, anatomical channel meaning, and event detection remain the caller's responsibility. The selected sessions must share stream names; feature summarization additionally checks feature names within each stream. Feature callbacks receive matrices, not recording metadata. For rate-dependent features, process contexts with their own rate settings before pooling; a single fixed-rate callback is inappropriate when session rates differ.

Operates on a PhysioCohort multi-subject container (package PhysioCohort), which is not among this package's dependencies, so the example cannot run offline and is not executed.

Value

A multimodal_cycles object accepted by summarizeCycleFeatures(). Canonical keys include subject, session, trial, side and cycle. The participant_id alias is derived from subject_id. cohort_design contains the source cohort's subject/session table; recordings contains one entry per selected subject/session/trial context, including clock and per-modality acquisition metadata. Cycle rows retain input order.

See Also

segmentMultimodalCycles(), summarizeCycleFeatures()

Examples

```
## Not run:
# `cohort` is a PhysioCohort object; see the PhysioCohort package.
segmentCohortCycles(cohort, cycles = cohort_cycles)

## End(Not run)
```

segmentMultimodalCycles *Segment matched cycles on native modality sampling grids*

Description

Applies explicit shared-clock intervals to multiple modalities from one participant and trial. All cycle rows and their identifiers are retained. Intervals are half-open: `[start_time, end_time)`. Recording coverage is `[origin, origin + n_samples / sampling_rate)`, treating each acquired sample as the start of a sample interval. No interpolation or endpoint extension is performed. Boundary roundoff is tolerated up to one millionth of a sample.

Usage

```
segmentMultimodalCycles(streams, cycles, incomplete = c("error", "retain"))
```

Arguments

`streams`	Uniquely named list of modalities. Each entry is a list with `data` (nonempty real numeric matrix, time by channels), `sampling_rate` (positive finite Hz), `start_time` (finite shared-clock seconds), and nonempty character scalars `participant_id` and `trial_id`. All entries must declare the same participant and trial.
`cycles`	Nonempty data.frame with character columns `participant_id`, `trial_id`, `side`, `cycle_id` and numeric `start_time`, `end_time` in seconds. Composite identifiers must be unique; participant/trial must match streams. Additional columns are retained. Missing boundaries are reported as incomplete cycles. The caller supplies event-derived boundaries; this function does not infer cycles, verify event meaning, or select channels anatomically from the side label.
`incomplete`	Either `"error"` (default) or `"retain"`. In retain mode, incomplete modality/cycle segments are `NULL`, with explicit reasons.

Value

A `multimodal_cycles` list with `cycles`, `segments` (modality then cycle row; each valid segment contains `data`, `time`, `sample_index`), `status` (keys, cycle row, modality, complete flag, reason and sample count), `complete_cycles`, `recordings` metadata and `key_columns`. Reasons are `ok`, `missing_boundary`, `invalid_interval`, `out_of_range`, `no_samples`, or `nonfinite_data`. Internal nonfinite samples invalidate a segment; they are never silently interpolated or dropped.

See Also

`summarizeCycleFeatures()`

Examples

```
s <- list(data = matrix(rep(2, 20), ncol = 1), sampling_rate = 10,
          start_time = 0, participant_id = "P1", trial_id = "T1")
c <- data.frame(participant_id = "P1", trial_id = "T1", side = "L",
                cycle_id = c("C1", "C2"), start_time = 0:1, end_time = 1:2)
z <- segmentMultimodalCycles(list(emg = s, motion = s), c)
z$status
```

`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)
```

---

sensoryOrganizationTest *Sensory Organization Test (SOT) equilibrium and sensory-ratio scoring*

Description

Computes NeuroCom-style Sensory Organization Test equilibrium scores, composite score, and the four sensory-analysis ratios from either per-trial equilibrium scores or per-trial peak-to-peak anteroposterior sway angles.

Usage

```
sensoryOrganizationTest(
  scores = NULL,
  sway_angle = NULL,
  theta_limit = 12.5,
  fall_as_zero = TRUE
)
```

Arguments

scores	A 6-row (conditions 1-6) numeric matrix / data.frame of equilibrium scores, columns = trials; or a list of 6 numeric vectors. NA entries (e.g. a fall) are treated as a fall with score 0 when fall_as_zero = TRUE.
sway_angle	Alternatively, peak-to-peak AP sway angles (degrees) in the same shape as scores; converted to equilibrium scores.
theta_limit	Theoretical AP stability limit in degrees (default 12.5).
fall_as_zero	Logical; treat NA trials as falls scored 0 (default TRUE).

Details

The equilibrium score for a trial is $EQ = 100 \times (\theta_{lim} - \theta_{pp})/\theta_{lim}$, where θ_{pp} is the peak-to-peak AP sway angle and θ_{lim} the theoretical AP stability limit (12.5 deg). The composite is the mean of the condition-1 and condition-2 average scores together with every individual trial score of conditions 3-6. The sensory ratios use condition average scores: SOM = C2/C1, VIS = C4/C1, VEST = C5/C1, PREF = (C3+C6)/(C2+C5).

Value

A `sot_result` object with per-trial `equilibrium`, per-condition `condition_means`, `composite`, and `ratios` (SOM/VIS/VEST/PREF).

References

Nashner LM (1997). "Computerized dynamic posturography."

See Also

`mCTSIB()`, `limitsOfStability()`, `swayMetrics()`

Examples

```
# Three trials per condition, equilibrium scores already computed
sc <- rbind(c(94, 95, 93), c(92, 91, 93), c(88, 90, 89),
            c(85, 84, 86), c(70, 72, 68), c(65, 66, 64))
sensoryOrganizationTest(sc)
```

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)
```

---

sparc *Spectral Arc Length (SPARC) movement smoothness*

Description

A duration- and amplitude-robust smoothness measure computed from the normalized Fourier magnitude spectrum of a speed profile. More negative = less smooth. A smooth minimum-jerk movement is approximately -1.4.

Usage

```
sparc(speed, fs, padlevel = 4, fc = 10, amp_th = 0.05)
```

Arguments

speed	Numeric speed (tangential velocity magnitude) profile.
fs	Sampling frequency in Hz.
padlevel	Zero-padding exponent added to the FFT length (default 4).
fc	Maximum cutoff frequency in Hz (default 10).
amp_th	Normalized amplitude threshold for the adaptive cutoff (0.05).

Value

The spectral arc length (negative scalar); NA if undefined.

References

Balasubramanian S, Melendez-Calderon A, Roby-Brami A, Burdet E (2015). "On the analysis of movement smoothness." J NeuroEng Rehabil 12:112.

Examples

```
t <- seq(0, 1, length.out = 200); tau <- t
speed <- 30 * tau^2 - 60 * tau^3 + 30 * tau^4  # minimum-jerk speed
sparc(speed, fs = 200)
```

---

stabilogramDiffusion *Stabilogram diffusion analysis (Collins & De Luca 1993)*

Description

Computes the mean square displacement (MSD) of the CoP as a function of the time interval and extracts the short-term and long-term diffusion coefficients and the critical point (the crossover between open-loop and closed-loop postural control). Planar (resultant) and per-axis analyses are returned.

Usage

```
stabilogramDiffusion(
  cop,
  sampling_rate,
  ap = NULL,
  ml = NULL,
  detrend = c("mean", "none"),
  max_interval = 10,
  short_max = 1,
  long_min = 2.5
)
```

Arguments

cop, ap, ml, sampling_rate, detrend	As in swayMetrics().
max_interval	Maximum time interval in seconds over which to compute the MSD (default 10, capped at half the record length).
short_max, long_min	Interval boundaries (seconds) for the short-term (default <= 1 s) and long-term (default >= 2.5 s) linear regions used to estimate the diffusion coefficients.

Value

A `stabilogram_diffusion` object with, per component (`planar`, `ap`, `ml`), the short/long diffusion coefficients, scaling exponents (Hurst) and the critical-point interval and MSD.

References

Collins JJ, De Luca CJ (1993). Exp Brain Res 95:308-318.

See Also

`swayMetrics()`

Examples

```
set.seed(1)
n <- 3000
cop <- data.frame(cop_x = cumsum(rnorm(n)) * 0.05,
                  cop_y = cumsum(rnorm(n)) * 0.05)
stabilogramDiffusion(cop, sampling_rate = 100)
```

---

staticOptimizationR *Pure-R static optimization of muscle activations*

Description

Distributes the net joint moment at each time frame across muscles by minimising the sum of squared activations (a quadratic muscle-effort criterion), subject to the moments being reproduced exactly and each activation staying within its bounds:

$$\min_{a} \sum_{i} w_i a_i^2 \quad \text{s.t.} \quad \tau_j = \sum_{i} a_i F_i^{max} r_{ji}, \; 0 \le a_i \le 1.$$

Each frame is an independent convex quadratic program (solved with **quadprog**), giving the unique minimum-effort activations. This mirrors the OpenSim Static Optimization tool (Anderson & Pandy 2001) and is used by `runStaticOptimization()` when OpenSim is unavailable.

Usage

```
staticOptimizationR(
  moment_arms,
  max_force,
  joint_moments,
  activation_bounds = c(0, 1),
  weights = NULL,
  passive_moments = NULL
)
```

Arguments

moment_arms	Muscle moment arms: either a single n_dof x n_muscle matrix r (moment arm of muscle *i* about degree-of-freedom *j*, reused for every frame) or a list of such matrices, one per frame.
max_force	Numeric vector of muscle maximum isometric forces (F_max, length n_muscle).
joint_moments	Net joint moments to reproduce: a length-n_dof vector (single frame) or an n_frames x n_dof matrix (one row per frame).
activation_bounds	Length-2 numeric c(lower, upper) activation bounds (default c(0, 1)).
weights	Optional length-n_muscle positive weights w_i for the effort criterion (default all 1).
passive_moments	Optional passive/gravity joint moments subtracted from joint_moments before optimization (same shape as joint_moments).

Value

A static_optimization_r object: a list with activations (n_frames x n_muscle), moments (the reproduced net joint moments, n_frames x n_dof), residual (max absolute moment error per frame), effort (sum of squared activations per frame), feasible (logical per frame), and the muscle / DOF counts.

References

Anderson FC, Pandy MG (2001). "Static and dynamic optimization solutions for gait are practically equivalent." J Biomech 34(2):153-161.

See Also

runStaticOptimization()

Examples

```
if (requireNamespace("quadprog", quietly = TRUE)) {
  # Two muscles with unit moment arm and F_max reproduce a moment of 30.
  r <- matrix(c(1, 1), nrow = 1)
  so <- staticOptimizationR(r, max_force = c(100, 100), joint_moments = 30)
  so$activations
}
```

`strapdownIntegrate` *ZUPT-aided strapdown integration of world-frame acceleration*

Description

Double-integrates world-frame linear acceleration to velocity and position, applying a zero-velocity update (ZUPT) at each stance sample: the integrated velocity is de-drifted so that it returns to zero at every stance sample (linear de-drifting within each swing), which bounds the per-stride integration drift.

Usage

```
strapdownIntegrate(accel_world, sampling_rate, stance)
```

Arguments

`accel_world`	Numeric matrix (n x 3) of gravity-free acceleration in the world frame (m/s^2), e.g. from `removeGravity()` rotated into world axes.
`sampling_rate`	Sampling rate in Hz.
`stance`	Logical vector of length n, `TRUE` at stance samples (from `detectStanceZUPT()`).

Value

A list with numeric matrices `velocity` and `position` (each n x 3).

References

Mariani B, Hoskovec C, Rochat S, Bula C, Penders J, Aminian K (2010). "3D gait assessment in young and elderly subjects using foot-worn inertial sensors." J Biomech 43(15):2999-3006.

See Also

`detectStanceZUPT()`, `footImuGait()`.

Examples

```
stance <- rep(c(TRUE, FALSE, TRUE, FALSE, TRUE), c(60, 40, 60, 40, 60))
accel_world <- matrix(0, length(stance), 3)
si <- strapdownIntegrate(accel_world, sampling_rate = 200, stance = stance)
names(si)
```

summarizeCycleFeatures *Summarize matched cycles into keyed feature blocks*

Description

Applies the selected modality's feature function to each native-grid segment and preserves its observation keys. A row is one cycle, not an independent participant. This function does not fit statistical models or enforce grouped train/test splits. Retain participant identifiers when building such analyses.

Usage

```
summarizeCycleFeatures(x, FUN, ..., incomplete = c("error", "exclude"))
```

Arguments

x	Result of segmentMultimodalCycles() or segmentCohortCycles().
FUN	Function, or a uniquely named list of functions matching modalities. Each function accepts a time-by-channel matrix and returns a nonempty, uniquely named, finite real numeric vector. Feature names must match across cycles within each modality; their order is reconciled by name. The function must handle the native grid appropriately; sampling rates are not equalized and samples are not weighted by duration automatically. Bind rate-dependent settings in per-modality functions using the recording metadata; metadata are not passed as implicit arguments to FUN.
...	Additional arguments passed to FUN.
incomplete	"error" (default) or explicit "exclude". Exclusion removes a cycle from all blocks if any modality was incomplete.

Value

List with numeric matrix blocks, matching per-modality key tables keys, retained cycle observations, all original status rows, cycle_rows and excluded_cycles (original cycle row indices), and recordings metadata. Blocks and keys can be passed to PhysioCrossModal's keyed block adapter. No invalid cycle is silently imputed or treated as a complete observation.

See Also

segmentMultimodalCycles()

Examples

```
# Given segmented cycles z and appropriately named matrix columns:
# features <- summarizeCycleFeatures(z, colMeans)
```

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
```

swayMetrics *Posturographic center-of-pressure sway metrics (Prieto 1996)*

Description

Computes the full Prieto et al. (1996) battery of postural-steadiness measures from a center-of-pressure (CoP) time series, together with the non-linear complexity descriptors used by schema_balance. Distances are reported in the units of the supplied CoP (typically mm or m); frequencies in Hz; areas in CoP-units squared; velocities in CoP-units per second.

Usage

```
swayMetrics(
  cop,
  sampling_rate,
  ap = NULL,
  ml = NULL,
  detrend = c("mean", "none"),
  entropy = TRUE,
  m = 2L,
  r = 0.2,
  base_of_support = NULL
)
```

Arguments

cop	Center-of-pressure data. A data.frame from calculateCOP() (columns cop_x = ML, cop_y = AP), a two-column matrix/data.frame (columns interpreted as ml, ap unless named), or NULL when supplying ap/ml directly.
sampling_rate	Sampling rate in Hz.
ap, ml	Optional anteroposterior / mediolateral CoP vectors (override cop).
detrend	Either "mean" (subtract the mean CoP, the Prieto convention) or "none".
entropy	Logical; compute per-axis sampleEntropy() and DFA-α (default TRUE).
m, r	Sample-entropy embedding dimension and tolerance (tolerance is a fraction of each axis SD).
base_of_support	Optional numeric c(ap = , ml =) giving the half extent of the base of support (in CoP units) about the mean CoP, used for time-to-boundary. NULL (default) returns NA for the TTB measures.

Details

The measures are defined relative to the mean CoP. With RD_n the resultant distance of sample n from the mean CoP and T the trial duration:

- **Path length** (total excursion, TOTEX): the summed point-to-point distance; **mean velocity** is TOTEX / T.
- **RMS distance** (RDIST) and **mean distance** (MDIST), resultant and per-axis.
- **95% confidence circle area** (AREA-CC) $= \pi(MDIST + z\, s_{RD})^2$ with $z = 1.645$.
- **95% confidence ellipse area** (AREA-CE) $= 2\pi F_{0.05[2,n-2]} \sqrt{s_{AP}^2 s_{ML}^2 - s_{AP,ML}^2}$.
- **Sway area** (AREA-SW), area swept per unit time.
- **Mean frequency** (MFREQ), the equivalent rotational frequency $MVELO/(2\pi\, MDIST)$.
- **Fractal dimension** (FD-CC) and frequency-domain measures (total power, 50%/95% power frequency, centroidal frequency, frequency dispersion).
- **Non-linear** sample entropy and DFA-α per axis, and optional time-to-boundary if a base of support is supplied.

Value

A `sway_metrics` object (a named list) with the Prieto measures and, in `$metrics`, a one-row `data.frame` aligned to the schema_balance metric names.

References

Prieto TE, Myklebust JB, Hoffmann RG, Lovett EG, Myklebust BM (1996). "Measures of postural steadiness." IEEE Trans Biomed Eng 43(9):956-966.

See Also

`calculateCOP()`, `analyzeForcePlate()`, `stabilogramDiffusion()`, `sensoryOrganizationTest()`, schema_balance

Examples

```
t <- seq(0, 30, by = 0.01)
cop <- data.frame(cop_x = 2 * sin(2 * pi * 0.2 * t),
                  cop_y = 3 * cos(2 * pi * 0.2 * t))
sm <- swayMetrics(cop, sampling_rate = 100)
sm$path_length
sm$mean_velocity
```

---

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 <- PhysioExperiment::PhysioExperiment(
  assays = S4Vectors::SimpleList(
    raw = matrix(rnorm(100), ncol = 1)
  ),
  colData = S4Vectors::DataFrame(label = "ch1", type = "marker"),
  samplingRate = 100
)
pe2 <- PhysioExperiment::PhysioExperiment(
  assays = S4Vectors::SimpleList(
    raw = matrix(rnorm(200), ncol = 1)
  ),
  colData = S4Vectors::DataFrame(label = "ch1", type = "marker"),
  samplingRate = 200
)
synced <- synchronizeSignals(list(pe1, pe2))
```

---

tardieuScore *Tardieu score: R1, R2 and the dynamic component*

Description

Combines a fast-velocity stretch (the catch angle R1) and a slow-velocity stretch (the full passive range R2) into the Tardieu dynamic component `R2 - R1` - the velocity-dependent (spastic) part of the range, distinct from a fixed contracture.

Usage

```
tardieuScore(fast, slow, sampling_rate = NULL, ...)
```

Arguments

fast	A tardieuStretch() result (fast stretch), or a raw fast-stretch angle trace (then scored with onset = "velocity").
slow	A tardieuStretch() result or a raw slow-stretch angle trace; R2 is its end-range angle (rom_max / max(angle)).
sampling_rate	Required only when fast is a raw angle trace.
...	Passed to tardieuStretch() when fast is raw.

Value

An S3 tardieu_score list: R1, R2, dynamic_component.

See Also

tardieuStretch(), reflexThreshold()

Examples

```
fs <- 200; t <- seq(0, 0.6, by = 1 / fs)
fast <- ifelse(t <= 0.25, 25 * t / 0.25, 25 + 5 * (t - 0.25) / 0.35)
slow <- 40 * t / 0.6
tardieuScore(tardieuStretch(fast, sampling_rate = fs, onset = "velocity"), slow)
```

---

tardieuStretch *Detect the spastic catch on a single passive stretch*

Description

Locates the Tardieu "catch" on one passive-stretch joint-angle trace and returns the catch angle (R1). The catch is detected from a reflex EMG burst (envelope crossing a baseline mean + threshold_sd * SD threshold), from a velocity arrest (angular velocity dropping below catch_fraction of its peak after the fastest stretch), or from both.

Usage

```
tardieuStretch(
  angle,
  emg = NULL,
  sampling_rate,
  emg_sampling_rate = sampling_rate,
  onset = c("emg", "velocity", "both"),
  baseline_sec = 0.5,
  threshold_sd = 3,
  catch_fraction = 0.5,
  envelope_ms = 50
)
```

Arguments

angle	Numeric joint-angle trace over the stretch (degrees; increasing as the joint is stretched).
emg	Optional EMG of the stretched muscle (numeric); enables the EMG reflex-onset trigger.
sampling_rate	Angle sampling rate (Hz).
emg_sampling_rate	EMG sampling rate (Hz); the EMG is resampled onto the angle time base with alignEMGtoMoCap(). Default = sampling_rate.
onset	Catch trigger: "emg", "velocity" or "both" (EMG-preferred, velocity fall-back).
baseline_sec	Baseline window (s) for the EMG threshold (default 0.5).
threshold_sd	EMG onset threshold in baseline SDs (default 3).
catch_fraction	Velocity-arrest fraction of peak velocity (default 0.5).
envelope_ms	EMG RMS-envelope window in ms (default 50).

Value

An S3 tardieu_stretch list: catch_angle (R1), catch_index, catch_velocity, peak_velocity, rom_max/rom_min, reflex_latency_ms (EMG onset relative to stretch onset), onset_method.

See Also

tardieuScore(), reflexThreshold()

Examples

```
fs <- 200; t <- seq(0, 0.6, by = 1 / fs)
angle <- ifelse(t <= 0.25, 25 * t / 0.25, 25 + 5 * (t - 0.25) / 0.35)
tardieuStretch(angle, sampling_rate = fs, onset = "velocity")$catch_angle
```

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"
)
```

`timeDelayEmbed` *Time-delay phase-space embedding*

Description

Reconstructs a scalar time series into a delay-embedded phase space $Y(i) = [x_i, x_{i+\tau}, \ldots, x_{i+(m-1)\tau}]$. When `delay` is `NULL` it is chosen at the first local minimum of the average mutual information (AMI); when `dim` is `NULL` it is chosen where the fraction of false nearest neighbours (FNN) first drops below `fnn_threshold`.

Usage

```
timeDelayEmbed(
  x,
  delay = NULL,
  dim = NULL,
  max_delay = 50L,
  max_dim = 10L,
  n_bins = 16L,
  fnn_threshold = 0.01
)
```

Arguments

`x`	Numeric time series.
`delay`	Embedding delay in samples; `NULL` to estimate via AMI.
`dim`	Embedding dimension; `NULL` to estimate via FNN.
`max_delay`	Maximum delay searched for the AMI minimum.
`max_dim`	Maximum embedding dimension searched for FNN.
`n_bins`	Histogram bins for the AMI estimate.
`fnn_threshold`	FNN fraction below which the dimension is accepted.

Value

A `time_delay_embedding` list with `embedded` (matrix, points x dim), `delay`, `dim`, `ami` (if estimated) and `fnn` (if estimated).

References

Fraser AM, Swinney HL (1986); Kennel MB, et al. (1992).

See Also

`maxLyapunovExponent()`, `sampleEntropy()`

Examples

```
x <- sin(seq(0, 20 * pi, length.out = 500))
emb <- timeDelayEmbed(x, delay = 5, dim = 3)
dim(emb$embedded)
```

---

timeToPeakTorque *Time to peak torque*

Description

Time to peak torque

Usage

```
timeToPeakTorque(
  torque,
  sampling_rate,
  onset = NULL,
  onset_method = c("absolute", "sd"),
  onset_threshold = 7.5,
  baseline_ms = 100,
  baseline_sd_k = 5
)
```

Arguments

`torque`	Numeric torque trace in N m.
`sampling_rate`	Sampling rate in Hz.
`onset`	Optional 1-based integer onset sample. When `NULL`, onset is detected from the baseline.
`onset_method`	Either a fixed `"absolute"` increase above baseline or a baseline `"sd"` rule.
`onset_threshold`	Absolute onset threshold in N m above baseline.
`baseline_ms`	Baseline duration in milliseconds.
`baseline_sd_k`	Number of baseline SDs used by the `"sd"` method.

Value

A named list containing time from onset, time from trace start, peak and onset indices, and peak torque.

Examples

```
torque <- c(rep(0, 100), seq(0.4, 120, by = 0.4), rep(120, 100))
timeToPeakTorque(torque, sampling_rate = 1000, onset_method = "sd")
```

---

timeToPeakVelocity *Time to peak reaching velocity*

Description

Time to peak reaching velocity

Usage

```
timeToPeakVelocity(
  speed,
  fs,
  onset_threshold = 0.05,
  threshold_type = c("relative", "absolute"),
  normalize = FALSE
)
```

Arguments

speed	Numeric non-negative tangential-speed profile.
fs	Sampling frequency in Hz.
onset_threshold	Non-negative threshold, interpreted according to threshold_type.
threshold_type	"relative" for a fraction of peak speed or "absolute" for speed units.
normalize	Logical; return time-to-peak divided by movement time instead of seconds.

Value

Time from detected onset to peak speed in seconds, or a unitless fraction of movement time when normalize = TRUE.

Examples

```
speed <- c(rep(0, 20), seq(0, 1, length.out = 30),
           seq(1, 0, length.out = 30), rep(0, 20))
timeToPeakVelocity(speed, fs = 100)
```

`toleranceNoiseCovariation` *Tolerance-Noise-Covariation (TNC) decomposition of result error*

Description

Decomposes the mean result error of a set of executions (Muller & Sternad 2004): how much error is removable by moving the mean to a more **tolerant** region, how much is due to dispersion (**noise**) around that region, and how much the observed inter-variable **covariation** already saves relative to a de-covaried (column-permuted) surrogate.

Usage

```
toleranceNoiseCovariation(
  execution,
  error_fn,
  optimum = NULL,
  n_surrogate = 200L
)
```

Arguments

`execution`	An `N x m` matrix of execution variables.
`error_fn`	A vectorised error function: given an `N x m` matrix it returns the length-`N` per-trial error (>= 0, 0 = perfect).
`optimum`	Length-`m` execution vector achieving (near-)zero error, used as the most tolerant target. When `NULL`, the dispersion (noise) reference is the cloud centroid.
`n_surrogate`	Number of column-permutation surrogates for the covariation estimate (default 200).

Value

a `tnc_result` list: `tolerance`, `noise`, `covariation` (error components; covariation > 0 = the observed covariation reduces error), `e_data` (mean data error), `e_optimum`.

References

Muller H, Sternad D (2004) J Exp Psychol Hum Percept Perform 30:212-233; Cohen RG, Sternad D (2009) Exp Brain Res 193:69-83.

See Also

`uncontrolledManifold()`

Examples

```
set.seed(3)
# redundant reaching task: result = x1 + x2, target 20, error squared
x1 <- rnorm(300, 12, 3); x2 <- 20 - x1 + rnorm(300, 0, 0.6)   # covary to hit 20
err <- function(M) (M[, 1] + M[, 2] - 20)^2
toleranceNoiseCovariation(cbind(x1, x2), err, optimum = c(10, 10))$covariation
```

`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

```
demo <- demoMoCapData(seed = 1)
pe_tracked <- trackMarkers(demo$mocap, method = "greedy")
pe_tracked
```

`tremorAmplitude` *Band-limited tremor amplitude (RMS)*

Description

Band-pass filters the signal to the tremor band and returns its root-mean- square amplitude (in the units of the input signal).

Usage

```
tremorAmplitude(signal, sampling_rate, band = c(3, 12), order = 4)
```

Arguments

signal	Numeric movement signal.
sampling_rate	Sampling rate in Hz.
band	Length-2 tremor band c(low, high) in Hz (default c(3, 12)).
order	Butterworth filter order (default 4).

Value

A list with rms and the band used.

See Also

tremorSpectrum(), tremorMetrics()

Examples

```
fs <- 100; t <- seq(0, 20, by = 1 / fs)
tremorAmplitude(2 * sin(2 * pi * 5 * t), fs)$rms   # ~ 2 / sqrt(2)
```

---

tremorMetrics *Tremor metrics (frequency + amplitude, with a task tag)*

Description

Combines tremorSpectrum() and tremorAmplitude() into a single tremor summary, tagged with the recording condition. The rest / postural / kinetic distinction is task-context metadata supplied by the caller (the segment recorded under that condition), not a separate algorithm.

Usage

```
tremorMetrics(
  signal,
  sampling_rate,
  condition = c("rest", "postural", "kinetic"),
  band = c(3, 12),
  ...
)
```

Arguments

signal	Numeric movement signal (one axis or magnitude).
sampling_rate	Sampling rate in Hz.
condition	Recording condition: "rest", "postural" or "kinetic".
band	Length-2 tremor band c(low, high) in Hz (default c(3, 12)).
...	Passed to tremorSpectrum().

Value

An S3 `tremor_metrics` list: `dominant_freq_hz`, `band_power_abs`, `band_power_rel`, `rms_amplitude`, `half_power_bw_hz`, `harmonic_ratio`, `condition`, `band`, `sampling_rate`.

See Also

`tremorSpectrum()`, `tremorAmplitude()`

Examples

```
fs <- 100; t <- seq(0, 20, by = 1 / fs)
tremorMetrics(1.5 * sin(2 * pi * 6 * t), fs, condition = "postural")
```

---

tremorSpectrum *Tremor power spectrum and dominant frequency*

Description

Estimates the power spectrum of a movement signal (Welch periodogram) and summarises the tremor within a frequency band: the dominant frequency, its peak power, absolute and relative in-band power, the half-power bandwidth (peak sharpness) and the first-harmonic ratio.

Usage

```
tremorSpectrum(
  signal,
  sampling_rate,
  band = c(3, 12),
  detrend = TRUE,
  seg_sec = NULL
)
```

Arguments

`signal`	Numeric vector: one axis, or the vector magnitude, of an accelerometer / gyroscope / displacement signal (gravity removed for raw accelerometry).
`sampling_rate`	Sampling rate in Hz.
`band`	Length-2 tremor band `c(low, high)` in Hz (default `c(3, 12)`).
`detrend`	Remove the signal mean before spectral estimation (default `TRUE`; each Welch segment is also demeaned).
`seg_sec`	Optional Welch segment length in seconds (default: about an eighth of the record, rounded to a power of two).

Value

A list: `dominant_freq_hz`, `peak_power`, `band_power_abs`, `band_power_rel` (in-band / total power), `half_power_bw_hz`, `harmonic_ratio` (power at 2*f0 / power at f0), and the `freq` / `psd` vectors.

See Also

tremorAmplitude(), tremorMetrics()

Examples

```
fs <- 100; t <- seq(0, 20, by = 1 / fs)
x <- 2 * sin(2 * pi * 5 * t)            # a 5 Hz, amplitude-2 tremor
tremorSpectrum(x, fs)$dominant_freq_hz
```

---

trunkCompensation *Trunk compensation during reaching*

Description

Decomposes hand transport into trunk translation and arm-relative transport, and optionally computes signed axial trunk rotation from shoulder markers.

Usage

```
trunkCompensation(
  trunk,
  hand,
  shoulder_r = NULL,
  shoulder_l = NULL,
  reach_axis = NULL,
  vertical_axis = 3L,
  onset = NULL,
  offset = NULL
)
```

Arguments

trunk, hand	Numeric n by 3 position matrices in a common coordinate frame and unit.
shoulder_r, shoulder_l	Optional right and left shoulder position matrices used for axial rotation. Supply both or neither.
reach_axis	Optional length-3 reach-direction vector. The net hand displacement is used by default.
vertical_axis	Index (1, 2, or 3) of the vertical coordinate axis.
onset, offset	Optional 1-based sample bounds.

Value

A trunk_compensation object containing trunk, hand, and arm transport, trunk contribution, signed rotation in degrees, and reach axis.

References

Cirstea MC, Levin MF (2000). Compensatory strategies for reaching in stroke. *Brain*, 123:940-953. doi:10.1093/brain/123.5.940

Examples

```
hand <- cbind(seq(0, 0.4, length.out = 20), 0, 0)
trunk <- cbind(seq(0, 0.1, length.out = 20), 0, 0)
trunkCompensation(trunk, hand)
```

`uncontrolledManifold` *Uncontrolled Manifold (UCM) analysis of motor variability*

Description

Decomposes trial-to-trial variance of the elemental variables into the part lying in the uncontrolled manifold (the null space of the task Jacobian – variability that does not change the task variable, "good") and the part orthogonal to it (variability that does, "bad"). A positive synergy index means the elements co-vary to stabilise the task.

Usage

```
uncontrolledManifold(theta, jacobian = NULL, task = NULL)
```

Arguments

`theta`	An `N x n` matrix: `N` trials (repetitions), `n` elemental variables (e.g. joint angles).
`jacobian`	The `d x n` task Jacobian at the mean configuration (`d` = dimension of the task variable). Provide this, or `task`.
`task`	Optional task function `R^n -> R^d`; its Jacobian is computed numerically at `colMeans(theta)` when `jacobian` is not given.

Value

a `ucm_result` list: `v_ucm`, `v_ort` (variance per DOF, parallel and orthogonal to the manifold), `v_total`, `delta_v` (synergy index, `(v_ucm - v_ort) / v_total`; > 0 = task-stabilising synergy), `n`, `d`, `N`.

References

Scholz JP, Schoner G (1999) Exp Brain Res 126:289-306; Latash ML, et al. (2002) Exerc Sport Sci Rev 30:26-31.

See Also

`goalEquivalentManifold()`, `toleranceNoiseCovariation()`

Examples

```
set.seed(1)
# redundant task: total = a1 + a2 held constant; variance mostly along the UCM
a1 <- rnorm(200, 30, 4); a2 <- 60 - a1 + rnorm(200, 0, 0.5)
uncontrolledManifold(cbind(a1, a2), jacobian = matrix(c(1, 1), nrow = 1))$delta_v
```

`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()

Examples

```
validateSchema(schema_gait)
```

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.

The result is unsigned, in $[0, 180]$: it says how far apart the two vectors are but not which side of v1 that v2 lies on, so mirror-image configurations (joint flexion versus hyperextension) are indistinguishable. For the signed variant, measured about a plane normal and therefore able to separate the two, use `calculateJointAngles(..., signed = TRUE)`.

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)
```

`vectorCoding` *Vector coding (coupling angle) between two joints*

Description

Vector coding quantifies coordination from the angle-angle diagram: the coupling angle is the orientation of the vector between successive points, and each frame is classified into in-phase (both joints rotating together), anti-phase (opposite), proximal-phase (`angle1` dominant) or distal-phase (`angle2` dominant) coordination (Chang et al. 2008; Needham et al. 2014).

Usage

```
vectorCoding(angle1, angle2)
```

Arguments

`angle1`, `angle2`	Numeric joint-angle series of equal length; `angle1` is the proximal joint (horizontal axis of the angle-angle plot).

Value

a `vector_coding` list: `coupling_angle` (degrees, 0-360, length n-1), `pattern` (a factor per frame) and `proportions` (time fraction per pattern).

References

Chang R, et al. (2008) J Biomech 41:3101-3105; Needham RA, et al. (2014) J Biomech 47:1235-1241.

See Also

`continuousRelativePhase()`, `coordinationVariability()`

Examples

```
t <- seq(0, 2 * pi, length.out = 101)
vectorCoding(sin(t), sin(t))$proportions           # all in-phase
```

`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()`

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE)) {
  set.seed(1)
  d <- matrix(rnorm(1000), 100, 10)
  ts <- waveformTSNE(d, perplexity = 5)
  dim(ts$embedding)
}
```

---

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

```
if (requireNamespace("uwot", quietly = TRUE)) {
  set.seed(1)
  d <- matrix(rnorm(1000), 100, 10)
  um <- waveformUMAP(d, n_neighbors = 5)
  dim(um$embedding)
}
```

`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

```
path <- tempfile(fileext = ".json")
writeBenchmarkManifest(path, n = 1, overwrite = TRUE)
file.exists(path)
```

writeC3D *Write a C3D Motion Capture File*

Description

Writes 3D marker trajectories (and optional analog channels) from a PhysioExperiment to a binary C3D file via the **c3dr** package. This is the inverse of readC3D(): a readC3D() -> writeC3D() -> readC3D() round-trip reproduces the marker coordinates (within 32-bit float precision) and preserves the point/analog sampling-rate ratio.

Usage

```
writeC3D(x, path, include_analog = TRUE)
```

Arguments

x	A PhysioExperiment with "position_x", "position_y", and "position_z" assays (as produced by readC3D() or readTRC()), or a MultiPhysioExperiment whose marker stream carries those assays.
path	Character string giving the output .c3d path.
include_analog	Logical; if TRUE (default) and the object carries analog data (in metadata(x)$analog_data, or as an "analog" stream of a MultiPhysioExperiment), that data is written as C3D analog channels. If FALSE, a marker-only C3D file is written.

Details

The point (marker) rate is taken from samplingRate(x). When analog data is written, the analog rate and the integer point/analog ratio (ANALOG:RATE / POINT:RATE, i.e. analog subframes per point frame) are derived from the analog block so the ratio round-trips exactly. Force platform metadata is not written (the marker and analog signals are preserved; force-plate corner/calibration parameters are dropped).

Value

The output path, invisibly.

References

C3D.org. "The C3D File Format." https://www.c3d.org/.

See Also

readC3D(), writeTRC(), writeMOT()

Examples

```
if (requireNamespace("c3dr", quietly = TRUE)) {
  pe <- readC3D(c3dr::c3d_example(), include_analog = TRUE)
  out <- tempfile(fileext = ".c3d")
  writeC3D(pe, out)
  pe2 <- readC3D(out)
}
```

`writeMOT` *Write an OpenSim Motion File (.mot)*

Description

Writes tabular time-series data (joint angles, kinematics, ...) from a `PhysioExperiment` to an OpenSim `.mot` file. This is the inverse of `readMOT()`: a `readMOT()` -> `writeMOT()` -> `readMOT()` round-trip reproduces every numeric column exactly and the derived sampling rate.

Usage

```
writeMOT(x, path)
```

Arguments

`x`	A `PhysioExperiment` whose first assay holds the data columns.
`path`	Character string giving the output `.mot` path.

Details

The time column is taken from `metadata(x)$time` when present, otherwise it is reconstructed from `samplingRate(x)`. The `inDegrees` header flag is preserved from `metadata(x)$inDegrees` (default `"yes"`).

Value

The output `path`, invisibly.

See Also

`readMOT()`, `writeTRC()`

Examples

```
mot_file <- system.file("testdata", "sample.mot", package = "PhysioMoCap")
if (nzchar(mot_file)) {
  pe <- readMOT(mot_file)
  out <- tempfile(fileext = ".mot")
  writeMOT(pe, out)
}
```

writeTRC *Write an OpenSim TRC File (.trc)*

Description

Writes 3D marker trajectories from a PhysioExperiment to an OpenSim .trc file. This is the inverse of readTRC(): a readTRC() -> writeTRC() -> readTRC() round-trip reproduces the marker coordinates exactly and the DataRate header field.

Usage

```
writeTRC(x, path)
```

Arguments

x	A PhysioExperiment with "position_x", "position_y", and "position_z" assays (as produced by readTRC() or readC3D()).
path	Character string giving the output .trc path.

Details

Marker labels, units (metadata(x)$Units, default "mm"), frame rate, and the time column (metadata(x)$time, else derived from samplingRate(x)) are preserved on write.

Value

The output path, invisibly.

See Also

readTRC(), writeMOT(), writeC3D()

Examples

```
trc_file <- system.file("testdata", "sample.trc", package = "PhysioMoCap")
if (nzchar(trc_file)) {
  pe <- readTRC(trc_file)
  out <- tempfile(fileext = ".trc")
  writeTRC(pe, out)
}
```

Index

∗ **datasets**
schema_balance, 253
schema_cutting, 253
schema_cycling, 254
schema_gait, 255
schema_jump, 255
schema_running, 256
schema_throw, 257

adlReachTask, 8
adlReachTask(), *139*
alignEMGtoMoCap, 9
alignEMGtoMoCap(), *113*, *276*
analyzeForcePlate, 11
analyzeForcePlate(), *13*, *24*, *36*, *38*, *56*, *85*, *196*, *218*, *272*
analyzeForcePlatePE, 12
analyzeForcePlatePE(), *11*, *56*, *196*, *197*
approx, *82*, *83*, *239*
approximateEntropy, 13
approximateEntropy(), *233*
assessMoCapReadiness, 14
assessMoCapReadiness(), *52*, *199*, *209*, *221*

batch_analyze_opensim, 15
batch_analyze_opensim(), *48*, *241*
batchNormalize, 17
batchNormalize(), *33*, *140*, *142*
benchmarkAgreement, 18
benchmarkAgreement(), *50*, *191*, *242*
benchmarkManifestTemplate, 19
benchmarkManifestTemplate(), *49*, *288*, *296*
blandAltman, *155*
blandAltman(), *18*
butterworthFilter, 19
butterworthFilter(), *85*, *86*, *115*, *138*, *250*

c3dEventTable, 21
c3dEventTable(), *215*
calculateCOM, 22
calculateCOM(), *24*, *30*, *55*, *56*, *261*
calculateCOP, 24
calculateCOP(), *11*, *45*, *271*, *272*
calculateGaitParameters, 25
calculateGaitParameters(), *31*, *55*, *56*, *163*, *168*, *198*, *270*, *273*
calculateJointAngles, 27, *120*
calculateJointAngles(), *78*, *106*, *121*, *185*, *207*, *290*
calculateSegmentCOM, 29
calculateSegmentCOM(), *23*, *261*
calculateStepSymmetry, 30
calculateStepSymmetry(), *26*, *179*, *273*
calibrateIMU, 31
calibrateIMU(), *76*, *236*
classifyGaitPatterns, 32
classifyGaitPatterns(), *53*, *58*, *61*, *63*, *93*, *151*
cohensD(), *161*
combineTrials, 33
combineTrials(), *17*
computeAcceleration, 34
computeAcceleration(), *36*, *40*, *41*, *64*
computeImpulse, 35
computeImpulse(), *38*, *43*, *105*
computeJerk, 36
computeJerk(), *35*
computeJointPower, 37
computeJointPower(), *115*, *118*, *121*, *122*, *124*
computeLoadingRate, 38
computeLoadingRate(), *11*, *36*, *105*
computeRMSEnvelope, 39
computeRMSEnvelope(), *140*, *203*, *232*
computeSpeed, 40, *135*
computeSpeed(), *8*, *41*, *210*
computeVelocity, 41, *135*
computeVelocity(), *35*, *36*, *40*, *64*, *170*, *210*

continuousRelativePhase, 42
continuousRelativePhase(), *44*, *291*
contractionWork, 43
coordinationVariability, 44
coordinationVariability(), *42*, *291*
copFromPressure, 45
correctSwaps, 46
correctSwaps(), *62*, *283*
create_schema_from_opensim, 47
create_schema_from_opensim(), *16*, *241*
createBenchmarkExample, 48
createBenchmarkExample(), *242*

defaultBenchmarkThresholds, 49
defaultBenchmarkThresholds(), *18*
define_skeleton, 50
define_skeleton(), *96, 97*, *220*, *227*, *264*
demoMoCapData, 51
demoMoCapData(), *15*, *209*
detectCircumduction, 52
detectCircumduction(), *93*
detectEvents, 53
detectEvents(), *48*, *56*, *79*, *131*, *262*
detectEventsZeni, 54
detectForcePlateContacts, 56
detectGaps, 57, *237*
detectGaps(), *82–84*, *237*
detectPushOffDeficit, 58
detectPushOffDeficit(), *93*
detectStanceZUPT, 59
detectStanceZUPT(), *88, 89*, *268*
detectStiffKnee, 60
detectStiffKnee(), *93*
detectSwaps, 62
detectSwaps(), *46*, *283*
detectTrendelenburg, 63
detectTrendelenburg(), *93*
differentiate, 64
differentiate(), *35, 36*, *41*, *250*
dimensionlessJerk, 65
dmpFit, 65
dmpFit(), *66, 67*, *206*
dmpGenerate, 66
dmpGenerate(), *65, 66*
downloadOpenCapModel, 67
downloadOpenCapModel(), *243–246*
dtwAverage, 68
dtwAverage(), *70*, *72, 73*
dtwClustering, 69
dtwClustering(), *69*, *72*, *159*
dtwDistance, 71
dtwDistance(), *69, 70*, *72, 73*, *159*
dtwDistanceMatrix, 72
dtwDistanceMatrix(), *69, 70*, *72*
dtwWarp, 73
dtwWarp(), *72*, *159*

endpointError, 74
endpointError(), *126*
estimateOrientation, 74
estimateOrientation(), *32*, *88, 89*, *123*, *130*, *236*
estimateSegmentInertia, 76
estimateSegmentInertia(), *115*, *117–119*, *252*, *261*
etaSquared(), *161*
eulerToQuaternion, 77
eulerToQuaternion(), *29*, *207*, *290*
Event, 78
Event(), *146*, *277*
extractPhase, 79
extractPhase(), *102*, *109*, *262*
extractWaveformFeatures, 80
extractWaveformFeatures(), *293–295*

fillGaps, 82
fillGaps(), *57*, *83, 84*, *237*
fillGapsLinear, 83
fillGapsLinear(), *57*, *82*, *84*, *237*
fillGapsSpline, 83
fillGapsSpline(), *82, 83*
filterGRF, 84
filterGRF(), *11*, *105*, *218*
filterSignals, 85
filterSignals(), *20*, *115*
floquetMultipliers, 87
floquetMultipliers(), *233*
footImuGait, 88
footImuGait(), *59, 60*, *110*, *268*
forwardDynamics, 89
forwardDynamics(), *154*, *187*
forwardKinematics2D, 90
forwardKinematics2D(), *120*, *251*
fPCA(), *161*

gaitAtaxiaIndex, 91
gaitAtaxiaIndex(), *126*
gaitDeviationIndex, 92

gaitDeviationIndex(), *95*, *151*
gaitPatternLibrary, 93
gaitPatternLibrary(), *33*, *61*
gaitProfileScore, 93
gaitProfileScore(), *92*, *95*, *134*
gaitVariableScore, 94
gaitVariableScore(), *92*, *94*, *134*
gdiBasis, 95
gdiBasis(), *92*
get_bone_connections, 96
get_bone_connections(), *50*, *98*, *264*
get_limb_pairs, 97
get_limb_pairs(), *50*, *98*
get_segment_lengths, 97
get_segment_lengths(), *96, 97*, *264*
getEvent, 98
getEvent(), *99, 100*
getEventNames, 99
getEventNames(), *99*, *102*
getPhase, 100
getPhase(), *99*, *101, 102*
getPhaseColors, 100
getPhaseColors(), *100*, *102*
getPhaseData, 101
getPhaseData(), *80*, *109*
getPhaseNames, 102
getPhaseNames(), *99–101*
getSchema, 103
getSchema(), *128*, *255*, *277*, *289*
goalEquivalentManifold, 103
goalEquivalentManifold(), *287*
grfLandmarks, 104
groodSuntayAngles, 105, *120*
groodSuntayAngles(), *28*, *121*

hamstringQuadRatio, 107
harmonicRatio, 108
hasValidPhases, 109
hasValidPhases(), *102*

instrumented10mWT, 110
instrumented10mWT(), *112*
instrumentedTUG, 111
instrumentedTUG(), *110*, *181*
integrateEMGMoCap, 112
integrateEMGMoCap(), *10*, *203*
inverseDynamics2D, 114
inverseDynamics2D(), *37*, *77*, *117–119*
inverseDynamics3D, 116
inverseDynamics3D(), *37*, *77*, *115*
inverseDynamicsRNE, 118
inverseDynamicsRNE(), *114, 115*, *117, 118*, *251, 252*
inverseKinematicsMarkers, 119
inverseKinematicsMarkers(), *90*, *251*

jointCoordinateSystem, *106*, 120
jointCoordinateSystem(), *27, 28*, *106*
jointWork, 121
jointWork(), *37*, *124*, *150*

kalmanOrientation, 122
kalmanOrientation(), *130*

labelPowerBursts, 124
labelPowerBursts(), *37*, *122*
ldlj, *65*, 125
limbAtaxiaIndex, 126
limbAtaxiaIndex(), *91*, *142*
limitsOfStability, 127
limitsOfStability(), *263*
listSchemas, 128
listSchemas(), *103*, *289*
localDynamicStability, 129
localDynamicStability(), *132*

mahonyAHRS, 130
mahonyAHRS(), *123*
manualEvents, 131
manualEvents(), *54*
maxLyapunovExponent, 132
maxLyapunovExponent(), *87*, *108*, *129*, *249*, *279*
mCTSIB, 133
mCTSIB(), *263*
movementAnalysisProfile, 134
movementAnalysisProfile(), *94, 95*, *165*
movementSmoothness, 135
movementTime, 136
movementUnits, 136
movementUnits(), *9*, *126*, *138, 139*
movingAverage, 137
movingAverage(), *86*, *250*

nhptDexterity, 138
normalizedTimeAxis, 139
normalizedTimeAxis(), *17*, *142*
normalizeEMG, 140

normalizeEMG(), *39*, *203*
normalizeMovement, 141
normalizeMovement(), *17*, *33*, *73*, *93, 94*, *134*, *140*, *262*

pathStraightness, 142
pathStraightness(), *126*
peakPressure, 143
peakTorque, 143
peakVelocity, 145
Phase, 145
Phase(), *79*, *277*
phaseDurations, 146
phaseDurations(), *147, 148*
phaseRatios, 147
phaseRatios(), *147, 148*
phaseTiming, 148
phaseTiming(), *80*, *147*, *262*
PhysioExperiment::blandAltman(), *156*
PhysioExperiment::samplingRate(), *211*
physioPalette, *155*
pipelineACLrts, 149
pipelineAmputee, 150
pipelineCPgait, 151
pipelinePDfog, 152
pipelineStrokePropulsion, 153
planarLimbModel, 154
planarLimbModel(), *90*
plotBlandAltman, 155
plotCorrelationMatrix, 156
plotCorrelationMatrix(), *161*
plotCycle, 157
plotCycle(), *164*, *166*, *168*
plotDTW, 159
plotEffectSizeForest, 160
plotEffectSizeForest(), *157*
plotFPCA, 161
plotGaitCycle, 162
plotGaitCycle(), *26*, *170*, *178*, *183*
plotGroupComparison, 163
plotGroupComparison(), *158*, *166*
plotMAP, 164
plotMAP(), *134*
plotMultiPanel, 165
plotMultiPanel(), *158*
plotPCAScatter, 166
plotPCAScatter(), *167*, *182*, *293*
plotPCAVariance, 167
plotPCAVariance(), *167*, *293*
plotPhaseDurations, 168
plotPhaseDurations(), *158*
plotPhasePortrait, 169
plotPhasePortrait(), *180*
plotPressureMap, 170
plotSkeleton, 171
plotSkeleton(), *173*, *175, 176*, *180*, *204*
plotSkeleton3D, 172
plotSkeleton3D(), *172*, *204*
plotSkeletonOverlay, 174
plotSkeletonOverlay(), *172*, *176*
plotSkeletonSequence, 175
plotSkeletonSequence(), *172*, *175*
plotSpaghetti, 177
plotSpaghetti(), *163*, *183*
plotSymmetry, 178
plotSymmetry(), *31*, *183*, *273*
plotTrajectory, 179
plotTUG, 180
plotUMAP, 181
plotUMAP(), *167*, *295*
plotWaveformComparison, 182
plotWaveformComparison(), *157*, *163, 164*, *178, 179*
poseFix, 183
poseFix(), *185, 186*
poseLengthCorrect, 185
poseLengthCorrect(), *184*
predictiveSimulation, 186
predictiveSimulation(), *90*, *154*
pressureAsymmetry, 187
pressureMovie, 188
pressureTimeIntegral, 189
print.ASFSkeleton, 190
print.ASFSkeleton(), *213*
print.benchmark_agreement, 191
print.benchmark_manifest_validation, 192
print.benchmark_manifest_validation(), *288*
print.benchmark_suite, 193
print.benchmark_suite(), *242*
print.detected_events, 194
print.dtw_clustering, 194
print.dtw_result, 195
print.forceplate_analysis, 195
print.forceplate_analysis(), *13*, *197*
print.forceplate_analysis_multi, 196

print.gait_parameters, 197
print.gait_parameters(), *270*
print.mocap_quickstart, 198
print.mocap_quickstart(), *209*
print.mocap_readiness, 199
print.mocap_readiness(), *15*
print.multi_trial_phases, 200
print.segmented_phases, 200
print.SkeletonModel, 201
print.waveform_pca, 202
print.waveform_umap, 202
processEMG, 203
processEMG(), *39*, *107*, *113*, *140*, *232*
projectTo2D, 204
projectTo2D(), *172, 173*
promCondition, 205
promCondition(), *206*
promFit, 205
promFit(), *66*, *205*

quaternionToEuler, 206
quaternionToEuler(), *29*, *76*, *78*, *236*, *290*
quickStartMoCap, 208
quickStartMoCap(), *15*, *52*, *198*

rateOfForceDevelopment, 209
reachingKinematics, 210
reachingKinematics(), *9*, *126*, *142*
readAMC, 212
readAMC(), *190*, *213*, *221*, *223*
readASF, 213
readASF(), *190*, *212*, *221*, *223*
readBVH, 214
readBVH(), *215*
readC3D, 215
readC3D(), *21, 22*, *214*, *221*, *225*, *230*, *297*, *299*
readDeepLabCut, 216
readDeepLabCut(), *220*, *227*
readGaitRec, 217
readMediaPipe, 219
readMediaPipe(), *184*, *217*, *227*
readMoCapAuto, 220
readMoCapAuto(), *223*, *231*
readMoCapCSV, 221
readMoCapCSV(), *212*, *218*, *221*, *231*
readMOT, 223, *224, 225*
readMOT(), *225*, *227*, *229, 230*, *298*
readOpenCap, *68*, 224
readOpenCap(), *68*, *217*, *224*, *229*, *246*
readOpenPose, 226
readOpenPose(), *183–185*, *214, 215*, *217*, *220*
readOpenSimOutputs, 227
readOpenSimOutputs(), *243, 244*
readPressureFrame, 228
readSTO, 229
readSTO(), *224*, *227*, *230*
readTRC, *224, 225*, 230
readTRC(), *214, 215*, *224, 225*, *227*, *229*, *297*, *299*
readVenus3D, 231
readVenus3D(), *283*
reconstructFPCA(), *161*
rectifyEMG, 232
rectifyEMG(), *39*, *140*, *203*
recurrenceQuantification, 233
recurrenceQuantification(), *14*, *87*
reflexThreshold, 234
reflexThreshold(), *275, 276*
regionalLoading, 235
removeGravity, 236
removeGravity(), *32*, *76*, *89*, *268*
reportGaps, 237
reportGaps(), *57*, *82*
resampleSignal, 238
resampleSignal(), *10*, *239*, *274*
resampleVector, 239
resampleVector(), *238*, *274*
run_opensim_toolchain, 240
run_opensim_toolchain(), *243, 244*
runBenchmarkSuite, 241
runBenchmarkSuite(), *18*, *49, 50*, *193*
runCMC, 242
runCMC(), *247, 248*
runOpenSimFromMarkers, *67*, 243
runOpenSimFromMarkers(), *68*, *245, 246*
runOpenSimFromOpenCap, *67*, 245
runOpenSimFromOpenCap(), *68*, *244*
runRRA, 246
runRRA(), *242*, *248*
runStaticOptimization, 247
runStaticOptimization(), *242*, *247*, *266, 267*

sampleEntropy, 248
sampleEntropy(), *14*, *271*, *279*
savgolFilter, 249

savgolFilter(), *20*, *64*, *86*, *138*
scaleBodyModel, 250
scaleBodyModel(), *115*, *119*, *252*
scalePlanarModel, 251
scalePlanarModel(), *120*
scaleSegmentModel, 252
scaleSegmentModel(), *251*
schema_balance, 253, *270*, *272*
schema_cutting, 253, *253*, *257*
schema_cycling, 254
schema_gait, *253*, *254*, 255, *256*, *257*
schema_jump, 255, *256*
schema_running, *254*, *255*, 256
schema_throw, *256*, 257
segmentCohortCycles, 257
segmentCohortCycles(), *22*, *269*
segmentMultimodalCycles, 259
segmentMultimodalCycles(), *257*, *258*, *269*
segmentParameters, 260
segmentParameters(), *22*, *23*, *30*, *77*
segmentPhases, 261
segmentPhases(), *33*, *54*, *80*, *102*, *109*, *122*, *131*, *140*, *142*, *146–148*
sensoryOrganizationTest, 262
sensoryOrganizationTest(), *128*, *133*, *272*
signal::butter(), *86*
signal::filtfilt(), *86*
SkeletonModel, 263
SkeletonModel(), *50*, *96–98*
smooth.spline, *82–84*
sparc, *135*, 264
sparc(), *9*, *126*, *211*
spline, *239*
stabilogramDiffusion, 265
stabilogramDiffusion(), *272*
staticOptimizationR, 266
staticOptimizationR(), *247*, *248*
stats::approx(), *10*
stats::filter(), *137*
stats::smooth.spline(), *184*
strapdownIntegrate, 268
strapdownIntegrate(), *59*, *60*, *89*
summarizeCycleFeatures, 269
summarizeCycleFeatures(), *258*, *259*
summarizeGaitParameters, 270
summarizeGaitParameters(), *26*, *91*, *198*
swayMetrics, 270
swayMetrics(), *45*, *91*, *127*, *128*, *133*, *263*, *265*, *266*
symmetryIndex, 272
symmetryIndex(), *23*, *31*, *179*
synchronizeSignals, 273
synchronizeSignals(), *113*, *238*, *239*

tardieuScore, 274
tardieuScore(), *234*, *276*
tardieuStretch, 275
tardieuStretch(), *234*, *275*
TaskSchema, 276
TaskSchema(), *54*, *79*, *99*, *101*, *103*, *128*, *131*, *146*, *253–257*, *289*
timeDelayEmbed, 278
timeDelayEmbed(), *132*
timeToPeakTorque, 279
timeToPeakVelocity, 280
toleranceNoiseCovariation, 281
toleranceNoiseCovariation(), *287*
trackMarkers, 282
trackMarkers(), *46*, *62*, *231*
tremorAmplitude, 283
tremorAmplitude(), *284–286*
tremorMetrics, 284
tremorMetrics(), *284*, *286*
tremorSpectrum, 285
tremorSpectrum(), *284*, *285*
trunkCompensation, 286

uncontrolledManifold, 287
uncontrolledManifold(), *104*, *281*

validateBenchmarkManifest, 288
validateBenchmarkManifest(), *19*, *192*, *296*
validateSchema, 289
validateSchema(), *103*, *128*, *277*
vectorAngle, 289
vectorAngle(), *29*, *78*, *207*
vectorCoding, 291
vectorCoding(), *42*, *44*

waveformPCA, 292
waveformPCA(), *81*, *167*, *182*, *294*, *295*
waveformTSNE, 293
waveformTSNE(), *81*, *295*
waveformUMAP, 294
waveformUMAP(), *81*, *182*, *293*, *294*
writeBenchmarkManifest, 296

writeBenchmarkManifest(), *19*
writeC3D, 297
writeC3D(), *299*
writeMOT, 298
writeMOT(), *297*, *299*
writeTRC, 299
writeTRC(), *297*, *298*