Package 'navigation'

May 9, 2026

Type Package

Title Analyze the Impact of Sensor Error Modelling on Navigation Performance

Version 0.0.1

LazyData true

SystemRequirements GNU make

LazyDataCompression xz

Maintainer Lionel Voirol <lionelvoirol@hotmail.com>

Description Implements the framework presented in Cucci, D. A., Voirol, L., Khaghani, M. and Guerrier, S. (2023) <doi:10.1109/TIM.2023.3267360> which allows to analyze the impact of sensor error modeling on the performance of integrated navigation (sensor fusion) based on inertial measurement unit (IMU), Global Positioning System (GPS), and barometer data.
The framework relies on Monte Carlo simulations in which a Vanilla Extended Kalman filter is coupled with realistic and user-configurable noise generation mechanisms to recover a reference trajectory from noisy measurements.
The evaluation of several statistical metrics of the solution, aggregated over hundreds of simulated realizations, provides reasonable estimates of the expected performances of the system in real-world conditions.

Depends R (>= 4.0.0), plotly, magrittr, simts

License AGPL-3

Imports expm, rbenchmark, leaflet, MASS, pbmcapply, Rcpp (>= 0.8.0), RcppArmadillo (>= 0.2.0)

RoxygenNote 7.2.3

Encoding UTF-8

Suggests knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/SMAC-Group/navigation

BugReports https://github.com/SMAC-Group/navigation/issues

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Davide A. Cucci [aut],
Lionel Voirol [aut, cre],
Mehran Khaghani [aut],
Stéphane Guerrier [aut]

Repository CRAN

Date/Publication 2023-05-16 17:50:02 UTC

Contents

`compute_coverage` *Compute Coverage*

Description

Compute Empirical Coverage

Usage

```
compute_coverage(
  sols,
  alpha = 0.95,
  step = 100,
  idx = 1:6,
  progressbar = FALSE
)
```

Arguments

sols	The set of solutions returned by the navigation function
alpha	size of the confidence interval
step	Step
idx	Components of the states to be considered (default: position and orientation)
progressbar	A boolean specifying whether or not to show a progress bar. Default is FALSE.

Value

Return a coverage.stat object which contains the empirical coverage of the navigation solutions.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# load data
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 20,
  freq.imu = 100,
  # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1,
  # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 10,
  # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 15
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
```

```
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "1",
  # order of the Taylor expansion of the matrix exponential
  # used to compute Phi and Q matrices
```

```
  compute_PhiQ_each_n = 20,
  # compute new Phi and Q matrices
  # every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
) # keep one covariance every second
coverage <- compute_coverage(res, alpha = 0.7, step = 100, idx = 1:6)
plot(coverage)
```

`compute_mean_orientation_err` *Compute mean orientation error*

Description

Compute the mean orientation error (|| log(A^T * B) ||)

Usage

```
compute_mean_orientation_err(sols, step = 1, t0 = NULL, tend = NULL)
```

Arguments

`sols`	The set of solutions returned by the `navigation` function
`step`	do it for one sample out of `step`
`t0`	Start time for RMS calculation (default: beginning)
`tend`	Start time for RMS calculation (default: end)

Value

Return a `navigation.stat` object which contains the mean orientation error over the fused trajectories.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# load data
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
timing <- make_timing(
  nav.start = 0,
  # time at which to begin filtering
  nav.end = 20,
  freq.imu = 100,
```

```
  # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1,
  # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 10,
  # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 15
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalman filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
```

```
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "1",
  # order of the Taylor expansion of the matrix exponential
  # used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
) # keep one covariance every second


oe <- compute_mean_orientation_err(res, step = 25)
plot(oe)
```

`compute_mean_position_err` *Compute mean position error*

Description

Compute the mean position error (norm of 3D NED error)

Usage

```
compute_mean_position_err(sols, step = 1, t0 = NULL, tend = NULL)
```

Arguments

`sols`	The set of solutions returned by the `navigation` function
`step`	do it for one sample out of `step`
`t0`	Start time for RMS calculation (default: beginning)
`tend`	Start time for RMS calculation (default: end)

Value

Return a `navigation.stat` object which contains the mean position error over the fused trajectories.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# load data
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
timing <- make_timing(
  nav.start = 0,
  # time at which to begin filtering
  nav.end = 20,
  freq.imu = 100,
  # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1,
  # GNSS frequency
  freq.baro = 1,
  # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 10,
  # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 15
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
```

```
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "2",
  # order of the Taylor expansion of the matrix exponential
  # used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps
  # (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
) # keep one covariance every second


pe <- compute_mean_position_err(res, step = 25)
plot(pe)
```

`compute_nees` *Compute Normalized Estimation Errror Squared (NEES)*

Description

Compute Normalized Estimation Errror Squared (NEES)

Usage

```
compute_nees(sols, step = 50, idx = 1:6, progressbar = FALSE)
```

Arguments

sols	The set of solutions returned by the navigation function
step	do it for one sample out of step
idx	Components of the states to be considered (default: position and orientation)
progressbar	A boolean specifying whether or not to show a progress bar. Default is FALSE.

Value

Return a nees.stat object which contains the Normalized Estimation Error Squared.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# load data
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned,
system = "ned")
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 30,
  freq.imu = 100,
  # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1,
  # GNSS frequency
  freq.baro = 1,
  # barometer frequency
  # (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 20,
  # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 25
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(name = "imu",
frequency = timing$freq.imu,
error_model1 = acc.mdl,
error_model2 = gyr.mdl)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
```

```
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl, error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "4",
  # order of the Taylor expansion of the matrix exponential
  #  used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps
  #  (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
) # keep one covariance every second
nees <- compute_nees(res, idx = 1:6, step = 100)
plot(nees)
```

example_1_traj_ellipsoidal

Example trajectory 1 in ellipsoidal coordinates

Description

Example trajectory 1 in ellipsoidal coordinates.

Usage

```
example_1_traj_ellipsoidal
```

Format

An object of class `matrix` (inherits from `array`) with 6000 rows and 7 columns.

example_1_traj_ned

Example trajectory 1 in NED coordinates

Description

Example trajectory 1 in NED coordinates.

Usage

```
example_1_traj_ned
```

Format

An object of class `matrix` (inherits from `array`) with 6000 rows and 7 columns.

example_2_traj_ellipsoidal

Example trajectory 2 in ellipsoidal coordinates

Description

Example trajectory 2 in ellipsoidal coordinates.

Usage

```
example_2_traj_ellipsoidal
```

Format

An object of class `matrix` (inherits from `array`) with 12001 rows and 7 columns.

example_2_traj_ned *Example trajectory 2 in NED coordinates*

Description

Example trajectory 2 in NED coordinates.

Usage

```
example_2_traj_ned
```

Format

An object of class `matrix` (inherits from `array`) with 12001 rows and 7 columns.

lemniscate_traj_ned *Lemniscate trajectory in NED coordinates*

Description

Lemniscate trajectory in NED coordinates.

Usage

```
lemniscate_traj_ned
```

Format

An object of class `matrix` (inherits from `array`) with 60001 rows and 7 columns.

make_sensor *Construct a* `sensor` *object*

Description

Construct a `sensor` object for IMU, GPS, and Baro from error model of class `ts.model`

Usage

```
make_sensor(
  name,
  frequency = 1,
  error_model1 = NULL,
  error_model2 = NULL,
  error_model3 = NULL,
  error_model4 = NULL,
  error_data1 = NULL
)
```

Arguments

name	Name of the sensor
frequency	Frequency associated with the error model
error_model1	Error model of class ts.model for either accelerometer (as part of imu), horizontal components of GPS position, or Barometer
error_model2	Error model of class ts.model for either gyroscope (as part of imu) or vertical component of GPS position
error_model3	Error model of class ts.model for horizontal components of GPS velocity
error_model4	Error model of class ts.model for vertical component of GPS velocity
error_data1	Vector of error observations.

Value

An object of class sensor containing sensor name and its additive error model along with the frequency associated to that model

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# IMU:
imu.freq <- 250
acc.mdl <- WN(sigma2 = 1.535466e-04) + RW(gamma2 = 1.619511e-10)
gyr.mdl <- WN(sigma2 = 1.711080e-03) + RW(gamma2 = 1.532765e-13)
imu.mdl <- make_sensor(
  name = "imu",
  frequency = imu.freq,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)

# GPS:
gps.freq <- 1
gps.mdl.pos.hor <- WN(sigma2 = 2^2)
```

```
gps.mdl.pos.ver <- WN(sigma2 = 4^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.04^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.06^2)
gps.mdl <- make_sensor(
  name = "gps", frequency = gps.freq,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)

# Baro:
baro.freq <- 1
baro.mdl <- WN(sigma2 = 0.5^2)
baro.mdl <- make_sensor(
  name = "baro",
  frequency = baro.freq,
  error_model1 = baro.mdl
)
```

`make_timing` *Construct a* `timing` *object*

Description

Construct a `timing` object controlling the timing and frequencies for navigation, making sure about the consistency and feasibility of provided information.

Usage

```
make_timing(
  nav.start = NULL,
  nav.end = NULL,
  freq.imu = NULL,
  freq.gps = NULL,
  freq.baro = NULL,
  gps.out.start = NULL,
  gps.out.end = NULL
)
```

Arguments

`nav.start`	Time at which navigation starts
`nav.end`	Time at which navigation ends
`freq.imu`	Frequency of generated IMU data (and hence that of navigation)
`freq.gps`	Frequency of generated GPS data
`freq.baro`	Frequency of generated Baro data

gps.out.start	Time at which GPS outage starts
gps.out.end	Time at which GPS outage ends

Value

An object of class timing containing sensor name and its additive error model along with the frequency associated to that model

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
timing <- make_timing(
  nav.start = 0,
  nav.end = 50,
  freq.imu = 10,
  freq.gps = 1,
  freq.baro = 1e-5,
  gps.out.start = 25.1,
  gps.out.end = 45
)
```

---

make_trajectory *Construct a* trajectory *object*

Description

Create a trajectory object from simple matrix input.

Usage

```
make_trajectory(
  data,
  system = "ellipsoidal",
  start_time = NULL,
  name = NULL,
  ...
)
```

Arguments

data	A multiple-column matrix. The first column corresponds to the measurment time (in seconds); columns 2, 3 and 4 corresponds to the positions (with the order lat, long and alt (in rad) if ellipsoidal coord or x_N, x_E and x_D for NED coord); columns 5, 6 and 7 (optional) corresponds to the attitude (with the order roll, pitch and yaw); columns 8, 9 and 10 (optional) corresponds to the velocity along the same axes are columns 2, 3 and 4.

system	A string corresponding to the coordinate system (possible choices: ellipsoidal or ned) considered.
start_time	A string (optional) corresponding to the start time for the trajectory.
name	A string (optional) corresponding to the name of the dataset.
...	Additional arguments.

Value

An object of class trajectory.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
n <- 100
dat <- cbind(
  seq(from = 0, to = 60 * 60, length.out = n),
  46.204391 * pi / 180 + cumsum(rnorm(n)) / 10^5,
  6.143158 * pi / 180 + cumsum(rnorm(n)) / 10^5,
  375 + cumsum(rnorm(n))
)
traj <- make_trajectory(data = dat, name = "My cool data")
traj
plot(traj)
```

---

navigation *Runs "IMU model evaluation" or "INS-GPS-Baro integrated navigation (sensor fusion)"*

Description

This function performs of the two following main tasks, base on the provided input. If a reference trajectory (traj.ref) is provided, it generates sensor data (IMU, GPS, Baro) corrupted by additive errors according to snsr.mdl, and performs navigation using KF.mdl as the sensor error model within the Kalman filter to evaluate how this particular model performs when navigating.

Usage

```
navigation(
  traj.ref,
  timing,
  snsr.mdl,
  KF.mdl,
  g = 9.8056,
  num.runs = 1,
```

```
  results.system = "ned",
  x_o = NULL,
  noProgressBar = FALSE,
  IC = NULL,
  imu_data = NULL,
  gps_data = NULL,
  baro_data = NULL,
  input.seed = 0,
  PhiQ_method = "exact",
  P_subsampling = 1,
  compute_PhiQ_each_n = 1,
  parallel.ncores = detectCores(all.tests = FALSE, logical = TRUE),
  tmpdir = tempdir()
)
```

Arguments

traj.ref	A trajectory object (see the documentation for make_trajectory), serving as the reference trajectory for generating sensor data and evaluating the error in navigation once performed. Only position and attitude data are required/considered, and velocity will be calculated from position.
timing	A timing object (see the documentation for make_timing) containing timing information such as start and end of navigation.
snsr.mdl	A sensor object (see the documentation for make_sensor) containing additive sensor error model to generate realistic sensor data.
KF.mdl	A sensor object (see the documentation for make_sensor) containing additive sensor error model to be used within the Kalman filter for navigation.
g	Gravitational acceleration.
num.runs	Number of times the sensor data generation and navigation is performed (Monte-Carlo simulation).
results.system	The coordinate system (ned/ellipsoidal) in which the results are reported (see the documentation for make_trajectory).
x_o	Origin of the fixed ned frame.
noProgressBar	A bolean specifying if there should not be a progress bar.
IC	Initial conditions. See the examples for the format.
imu_data	IMU data. See the examples for the format.
gps_data	GPS data. See the examples for the format.
baro_data	Baro data. See the examples for the format.
input.seed	Seed for the random number generator. Actual seed is computed as input.seed * num.runs + run
PhiQ_method	String that specify the method to compute Phi and Q matrices, can be "exact" or the order of the Taylor expansions to use.
P_subsampling	(memory optimization) store only one sample of the P matrix each P_subsampling time instants.

`compute_PhiQ_each_n`
: Specify the interval of IMU measurements between each computation of PhiQ.

`parallel.ncores`
: The number of cores to be used for parallel Monte-Carlo runs.

`tmpdir`
: Where to store temporary navigation output. It should not be mapped on a filesystem which lives in RAM.

Value

An object of `navigation` class containing the reference trajectory, fused trajectory, sensor data, covariance matrix, and time.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# load data
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 15,
  freq.imu = 100, # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1, # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 8, # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 13
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
```

```
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "1",
  # order of the Taylor expansion of the matrix exponential
  # used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
) # keep one covariance every second
```

`plot.coverage.stat` *Plot multiple* `coverage.stat` *objects*

Description

plot multiple coverages alltogether

Usage

```
## S3 method for class 'coverage.stat'
plot(..., legend = NA, title = NA)
```

Arguments

...	coverage, e.g., computed with compute_coverage
legend	Legend of the plot.
title	Title of the plot.

Value

a plot of the empirical coverage.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
plot(traj)
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 30,
  freq.imu = 100, # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1, # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 20, # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 25
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
```

```
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "2",
 # order of the Taylor expansion of the matrix exponential used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
)
# Empirical coverage
coverage <- compute_coverage(res, alpha = 0.7, step = 100, idx = 1:6)
plot(coverage)
```

---

plot.navigation *Plot a* navigation *object*

Description

This function enables the visualization of a navigation object, both in 2d and in 3d. The function therefore enables the comparison of the true trajectory with emulated trajectories. One can also plot the analysis of the error of the trajectories by comparing the L2 norm of the difference between emulated trajectories and the true trajectory over time.

Usage

```
## S3 method for class 'navigation'
plot(
  x,
  true_col = "#2980b9",
  col_fused_trans = "#EA5D0073",
  col_fused_full = "#EA5D00FF",
  plot_mean_traj = TRUE,
  plot_baro = TRUE,
  baro_col = "black",
  emu_to_plot = 1,
  plot3d = FALSE,
  plot_CI = FALSE,
  time_interval = 5,
  col_50 = "#E74C3C4D",
  col_95 = "#F5B0414D",
  col_50_brd = "#E74C3C",
  col_95_brd = "#F5B041",
  error_analysis = FALSE,
  emu_for_covmat = 1,
  nsim = 1000,
  col_traj_error = "#1C12F54D",
  time_interval_simu = 0.5,
  seed = 123,
  ...
)
```

Arguments

x	A navigation object
true_col	The color of the true trajectory
col_fused_trans	The color of the emulated trajectories
col_fused_full	The color of the mean trajectory of the emulated trajectories
plot_mean_traj	A Boolean indicating whether or not to plot the mean mean trajectory of the emulated trajectories. Default is True
plot_baro	A Boolean indicating whether or not to plot the barometer datapoint in tha Up coordinates plot. Default is True
baro_col	The color of the barometer datapoints

emu_to_plot	The emulated trajectory for which to plot confidence ellipses on the North-East coordinates plot
plot3d	A Boolean indicating whether or not to plot the 3d plot of the trajectory
plot_CI	A Boolean indicating whether or not to plot the confidence intervals for both 2d plots
time_interval	A value in seconds indicating the interval at which to plot the CI on the North-East coordinates plot
col_50	The color for the 50% confidence intervals.
col_95	The color for the 95% confidence intervals.
col_50_brd	The color for the 50% confidence intervals borders.
col_95_brd	The color for the 95% confidence intervals.
error_analysis	A Boolean indicating whether or not to display an error analysis plot of the emulated trajectories
emu_for_covmat	The emulated trajectory for which to use the var-cov matrix in order to simulate data and compute the CI of the error
nsim	An integer indicating the number of trajectories simulated in order to compute the CI
col_traj_error	The color for the trajectory estimation error
time_interval_simu	time interval simu
seed	A seed for plotting
...	additional plotting argument

Value

A 2D or 3D plot of the trajectory with the fused trajectories.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
plot(traj)
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 20,
  freq.imu = 100, # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1, # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 5, # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 15
)
```

```
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 1 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
```

```
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "3",
 # order of the Taylor expansion of the matrix exponential used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
)
plot(res)
# 3D plot
plot(res, plot3d = TRUE)
plot(res, error_analysis = TRUE)
```

`plot.navigation.stat` *Plot multiple* `navigation.stat` *objects*

Description

plot multiple stats alltogether

Usage

```
## S3 method for class 'navigation.stat'
plot(..., legend = NA, title = NA, xlim = c(NA, NA), ylim = c(NA, NA))
```

Arguments

`...`	navigation statistics, e.g., computed with `compute_mean_position_err`
`legend`	The legend
`title`	The title
`xlim`	xlim
`ylim`	ylim

Value

A plot of the position or orientation error.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
plot(traj)
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 15,
  freq.imu = 100, # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1, # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 8, # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 13
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
```

```
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 1 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "2",
 # order of the Taylor expansion of the matrix exponential used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
)
# Mean orientation error
pe <- compute_mean_position_err(res, step = 25)
plot(pe)
```

`plot.nees.stat` *Plot multiple* `nees.stat` *objects*

Description

plot multiple `nees.stat` objects alltogether

Usage

```
## S3 method for class 'nees.stat'
plot(..., alpha = 0.95, legend = NA, title = NA)
```

Arguments

`...`	NEES, e.g., computed with `compute_nees`
`alpha`	for the confidence interval plot
`legend`	legend of the plot.
`title`	title of the plot.

Value

Produce a plot of the Normalized Estimation Error Squared (NEES) for the `nees.stat` object provided.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned, system = "ned")
plot(traj)
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 30,
  freq.imu = 100, # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1, # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 20, # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 25
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
```

```
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 2 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "1",
 # order of the Taylor expansion of the matrix exponential used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
)

nees <- compute_nees(res, idx = 1:6, step = 100)
plot(nees)
```

`plot.trajectory` *Plot a* `trajectory` *object*

Description

Plot a `trajectory` object in 2D or 3D.

Usage

```
## S3 method for class 'trajectory'
```

```
plot(
  x,
  threeD = FALSE,
  col = "#2980b9",
  col_start = "#e67e22",
  col_end = "#e67e22",
  pch_points_start = 15,
  pch_points_end = 16,
  cex_points = 1.5,
  add_altitude = TRUE,
  n_split = 6,
  plot_end_points = TRUE,
  add_title = TRUE,
  threeD_line_width = 4,
  threeD_line_color = "#008080",
  threeD_col_grad = FALSE,
  threeD_grad_start = "#008080",
  threeD_grad_end = "#ab53cf",
  ...
)
```

Arguments

x	A trajectory object
threeD	A boolean indicating whether the plot should be 3D or 2D (default FALSE, 2D).
col	A string corresponding to the color of the line used for 2D trajectory (default "blue4").
col_start	A string corresponding to the color of the point used to denote the beginning of a 2D trajectory (default "green3").
col_end	A string corresponding to the color of the point used to denote the end of a 2D trajectory (default "red2").
pch_points_start	A numeric corresponding to the symbol (pch) of the points used to denote the beginning of a 2D trajectory (default 15).
pch_points_end	A numeric corresponding to the symbol (pch) of the points used to denote the end of a 2D trajectory (default 16).
cex_points	A numeric corresponding to the size (cex) of the points used to denote the beginning and the end of a 2D trajectory (default 1.5).
add_altitude	A boolean to indicate if the altitude should be plotted in 2D trajectory in NED system (default TRUE; altitude is plotted).
n_split	A numeric for the number of ticks in 2D plot with altitude profile, if NULL no ticks are added (default = 6).
plot_end_points	A boolean to indicate if points should be plotted at the beginning and the end of a 2D trajectory (default TRUE; points are plotted).

add_title
: A boolean or string. If a boolean is used it indicates if a title should be added to 2D trajectory (only active if name of trajectory exist); if a string is used it corresponds to the title (default TRUE).

threeD_line_width
: A numeric corresponding to the width of the line for a 3D trajectory (default 4).

threeD_line_color
: A string corresponding to the hex color code of the line used for a 3D trajectory (default "#008080").

threeD_col_grad
: A boolean to indicate if a color gradient should be used for a 3D trajectory (default FALSE).

threeD_grad_start
: A string corresponding to the hex color code for the start of the gradient (default "#008080").

threeD_grad_end
: A string corresponding to the hex color code for the end of the gradient (default "#ab53cf").

...
: Additional arguments affecting the plot produced.

Value

A trajectory plot.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
n <- 100
set.seed(123)
dat <- cbind(
  seq(from = 0, to = 60 * 60, length.out = n),
  46.204391 * pi / 180 + cumsum(rnorm(n)) / 10^5,
  6.143158 * pi / 180 + cumsum(rnorm(n)) / 10^5,
  375 + cumsum(rnorm(n))
)
traj <- make_trajectory(data = dat, name = "My cool data")
plot(traj)
plot(traj, threeD = TRUE)
plot(traj,
  threeD = TRUE, threeD_line_width = 8,
  threeD_line_color = "#e74c3c"
)
plot(traj,
  threeD = TRUE,
  threeD_col_grad = TRUE
)
plot(traj,
  threeD = TRUE, threeD_col_grad = TRUE,
```

```
  threeD_grad_start = "#e74c3c",
  threeD_grad_end = "#d68910"
)

traj <- make_trajectory(data = dat, name = "My cool data", system = "ned")
plot(traj)
plot(traj, col = "orange2", col_start = "pink", col_end = "purple")
plot(traj, pch_points_start = 15, cex_points = 3)
plot(traj, plot_end_points = FALSE)
plot(traj, plot_end_points = FALSE, add_title = FALSE)
```

`plot_imu_err_with_cov` *Plot IMU error with covariances*

Description

this function plots the estimated IMU errors with covariance of a solution computed with the `navigation` function

Usage

```
plot_imu_err_with_cov(sol, idx = 1, error = TRUE, step = 10)
```

Arguments

`sol`	The set of solutions returned by the `navigation` function
`idx`	Which Monte-Carlo solution to plot
`error`	Whether to plot the error with respect to the reference or the estimated values
`step`	Plot one time out of `step`

Value

A plot of the estimated IMU errors with covariance.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned,
 system = "ned")
plot(traj)
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 20,
```

```
  freq.imu = 100,
  # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1,
  # GNSS frequency
  freq.baro = 1,
  # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 10, # to simulate a GNSS outage, set a time before nav.end
  gps.out.end =15
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
```

```
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "2",
 # order of the Taylor expansion of the matrix exponential used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
)
plot_imu_err_with_cov(res, error=FALSE)
```

`plot_nav_states_with_cov` *Plot navigation states with covariance*

Description

this function plots the navigation states with estimated covariance of a solution computed with the `navigation` function

Usage

```
plot_nav_states_with_cov(sol, idx = 1, cov_idx = 1, error = TRUE, step = 10)
```

Arguments

`sol`	The set of solutions returned by the `navigation` function
`idx`	Which Monte-Carlo solution to plot (can be a vector)
`cov_idx`	Which Monte-Carlo solution to use for confidence intervals
`error`	Wether to plot the error with respect to the refefence or the estimated values
`step`	Plot one time out of `step`

Value

a plot of the navigation states with the estimated covariance.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("lemniscate_traj_ned")
head(lemniscate_traj_ned)
traj <- make_trajectory(data = lemniscate_traj_ned,
 system = "ned")
plot(traj)
timing <- make_timing(
  nav.start = 0, # time at which to begin filtering
  nav.end = 20,
  freq.imu = 100,
  # frequency of the IMU, can be slower wrt trajectory frequency
  freq.gps = 1, # GNSS frequency
  freq.baro = 1,
  # barometer frequency (to disable, put it very low, e.g. 1e-5)
  gps.out.start = 10,
  # to simulate a GNSS outage, set a time before nav.end
  gps.out.end = 15
)
# create sensor for noise data generation
snsr.mdl <- list()
# this uses a model for noise data generation
acc.mdl <- WN(sigma2 = 5.989778e-05) +
  AR1(phi = 9.982454e-01, sigma2 = 1.848297e-10) +
  AR1(phi = 9.999121e-01, sigma2 = 2.435414e-11) +
  AR1(phi = 9.999998e-01, sigma2 = 1.026718e-12)
gyr.mdl <- WN(sigma2 = 1.503793e-06) +
  AR1(phi = 9.968999e-01, sigma2 = 2.428980e-11) +
  AR1(phi = 9.999001e-01, sigma2 = 1.238142e-12)
snsr.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
# RTK-like GNSS
gps.mdl.pos.hor <- WN(sigma2 = 0.025^2)
gps.mdl.pos.ver <- WN(sigma2 = 0.05^2)
gps.mdl.vel.hor <- WN(sigma2 = 0.01^2)
gps.mdl.vel.ver <- WN(sigma2 = 0.02^2)
snsr.mdl$gps <- make_sensor(
  name = "gps",
  frequency = timing$freq.gps,
  error_model1 = gps.mdl.pos.hor,
  error_model2 = gps.mdl.pos.ver,
  error_model3 = gps.mdl.vel.hor,
  error_model4 = gps.mdl.vel.ver
)
# Barometer
```

```
baro.mdl <- WN(sigma2 = 0.5^2)
snsr.mdl$baro <- make_sensor(
  name = "baro",
  frequency = timing$freq.baro,
  error_model1 = baro.mdl
)
# define sensor for Kalmna filter
KF.mdl <- list()
# make IMU sensor
KF.mdl$imu <- make_sensor(
  name = "imu",
  frequency = timing$freq.imu,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
KF.mdl$gps <- snsr.mdl$gps
KF.mdl$baro <- snsr.mdl$baro
# perform navigation simulation
num.runs <- 5 # number of Monte-Carlo simulations
res <- navigation(
  traj.ref = traj,
  timing = timing,
  snsr.mdl = snsr.mdl,
  KF.mdl = KF.mdl,
  num.runs = num.runs,
  noProgressBar = TRUE,
  PhiQ_method = "2",
 # order of the Taylor expansion of the matrix exponential used to compute Phi and Q matrices
  compute_PhiQ_each_n = 10,
  # compute new Phi and Q matrices every n IMU steps (execution time optimization)
  parallel.ncores = 1,
  P_subsampling = timing$freq.imu
)
plot_nav_states_with_cov(res, idx = 1:5, error = TRUE)
```

`print.sensor` *Print a* `sensor` *object parameters (name, frequency and error model)*

Description

Print method for a `sensor` object

Usage

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

Arguments

x	A sensor object.
...	Further arguments passed to or from other methods.

Value

Print the sensor object name and specifications in the console.

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
# IMU:
imu.freq <- 250
acc.mdl <- WN(sigma2 = 1.535466e-04) + RW(gamma2 = 1.619511e-10)
gyr.mdl <- WN(sigma2 = 1.711080e-03) + RW(gamma2 = 1.532765e-13)
imu.mdl <- make_sensor(
  name = "imu",
  frequency = imu.freq,
  error_model1 = acc.mdl,
  error_model2 = gyr.mdl
)
print(imu.mdl)
```

X_ellips2ned *Transform position from ellipsoidal to NED coordinates*

Description

Transform position from ellipsoidal coordinates to a fixed Cartesian NED frame

Usage

```
X_ellips2ned(x, x_o = NULL)
```

Arguments

x	An object of class trajectory in "ellipsoidal" system or a matrix of position data with latitude, longitude, and altitude
x_o	Origin of the fixed Cartesian NED frame expressed in ellipsoidal coordinates

Value

An object of class trajectory in "NED" system or a matrix of position data with x_N, x_E, and x_D, according to the type of input x

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
library(navigation)
data("example_1_traj_ellipsoidal")
traj_ellips <- make_trajectory(example_1_traj_ellipsoidal, system = "ellipsoidal")
plot(traj_ellips)
plot(traj_ellips, threeD = TRUE)
traj_ned <- X_ellips2ned(traj_ellips, x_o = example_1_traj_ellipsoidal[1, -1])
plot(traj_ned)
```

X_ned2ellips *Transform position from NED to ellipsoidal coordinates*

Description

Transform position from a fixed Cartesian NED frame to ellipsoidal coordinates

Usage

```
X_ned2ellips(x, x_o = NULL)
```

Arguments

x	An object of class `trajectory` in "NED" system or a matrix of position data with x_N, x_E, and x_D
x_o	Origin of the fixed Cartesian NED frame expressed in ellipsoidal coordinates

Value

An object of class `trajectory` in "ellipsoidal" system or a matrix of position data with latitude, longitude, and altitude, according to the type of input `x`

Author(s)

Davide Cucci, Lionel Voirol, Mehran Khaghani, Stéphane Guerrier

Examples

```
data("example_1_traj_ned")
traj_ned <- make_trajectory(example_1_traj_ned, system = "ned")
plot(traj_ned)
traj_ellips <- X_ned2ellips(traj_ned, x_o = example_1_traj_ellipsoidal[1, -1])
plot(traj_ellips, threeD = FALSE)
plot(traj_ellips, threeD = TRUE)
```

Index