Package ‘smlmkalman’

November 15, 2022

Type Package

Title Generation and Tracking of Super-Resolution Filamentous Datasets

Version 0.1.1

Author Andrew Buist [aut, cre]

Maintainer Andrew Buist <andrew.buist99@outlook.com>

Depends R (>= 4.1.0), stats, spdep, pracma, scales, truncnorm

Description A pair of functions that allow for the generation and tracking of coordinate data clouds without a time dimension,
primarily for use in super-resolution plant micro-tubule image segmentation.

License GPL-2

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2022-11-15 21:10:02 UTC

R topics documented:

`crescent_kf` *crescent_kf*

Description

A two-dimensional Kalman Filter for use in image segmentation of filamentous structures. Uses a novel "crescent sampling" method to generate the mean z value from local conditions, so time domain can be inferred from local point distribution.

Usage

```
crescent_kf(x, p_length,
            x_hat_s, sigma_s,
            B_s, d,
            Q, R,
            alpha, beta, gamma,
            overwrite, verbose)
```

Arguments

x	The dataset to be tracked - a two-column matrix or dataframe of x and y values.
p_length	The maximum number of predictions to be made.
x_hat_s	A vector containing two elements: the initial x- and y-values of the tracking agent.
sigma_s	A vector containing four elements: the initial error covariance matrix of the tracking agent (typically in the format c(x, 0, 0, y), where x and y are the error in x and y, and the non-diagonal elements are assumed to be 0)
B_s	A value from 0 to 360 indicating the initial angle of trajectory of the tracking agent in degrees (0 = east, 90 = north, 180 = west, 270 = south)
d	A value indicating the "step-length" of the Kalman Filter in the B direction at each timestep.
Q	A vector containing four elements: the process noise covariance matrix (typically in the format c(x, 0, 0, y), where x and y are the noise in x and y, and the non-diagonal elements are assumed to be 0)
R	A vector containing four elements: the measurement noise covariance matrix (typically in the format c(x, 0, 0, y), where x and y are the noise in x and y, and the non-diagonal elements are assumed to be 0)
alpha	A value indicating the multiplier for the expansion of the error covariance matrix during z sampling.
beta	A value indicating the minimum number of datapoints in the z sampling crescents which validates the timestep.
gamma	A value indicating the maximum number of rounds of alpha expansion of the error covariance matrix during z sampling (if gamma is reached before the crescent contains datapoints >= beta, the function exits prematurely)
overwrite	A TRUE/FALSE statement which dictates whether or not, after z sampling, the error covariance matrix should be overwritten by the z sampling matrix (FALSE by default, when TRUE may cause unexpected errors/tracking paths!)
verbose	A TRUE/FALSE statement that dictates whether the current place in the function loop should be printed to the terminal.

Value

A list containing four elements:

x_hat	A two-column dataframe of x- and y-values predicted by the Kalman Filter.

`sigma` A two-column dataframe of covariance error in x and y.

`search` A two-column dataframe of z search radius in x and y.

`allocated` A three-column dataframe, with the first two columns being equal to the input data (x), and the third column being a numeric ID of unobserved (allocated[,3] = 0) and observed (allocated[,3] >= 1) value, where the value of the ID indicates at which timestep they were observed.

Author(s)

Andrew Buist

Examples

```
#Generate loop-de-loop spline
data = as.data.frame(matrix(nrow = 1000, ncol = 3))
data[1:150,3] = 0
data[151:420,3] = c(1:270)
data[421:580,3] = 270
data[581:850,3] = c(270:1)
data[851:1000,3] = 0

data[1,1:2] = c(1,0)

library(spdep)
for(i in 2:1000){
  data[i,1:2] = c(data[(i-1),1] + 1, data[(i-1),2])
  angle =  (data[i,3]*(pi/180))
 data[i,1:2] = (Rotation(as.matrix(data[i,1:2] - data[(i-1),1:2], nrow = 1, ncol = 2), angle)
  + data[(i-1),1:2])
}

#Randomly generate data around spline
data_loopy = as.data.frame(matrix(nrow = 10000, ncol = 2))
for(i in 1:1000){
  data_loopy[((i-1)*10 + 1):(i*10),1] = rnorm(10, data[i,1],5)
  data_loopy[((i-1)*10 + 1):(i*10),2] = rnorm(10, data[i,2],5)
}

#Plot randomly generated data
plot(data_loopy, cex = 0.1)
#Add spline line in red
lines(data[,1:2], col = "red", lwd = 2)

#Peform Kalman Filtering
test =  crescent_kf(x = data_loopy, p_length = 1000, x_hat_s = c(data_loopy[1,1],data_loopy[1,2]),
                    sigma_s = c(10,0,0,10), B_s = 0, d = 0.75,
                    alpha = 1.1, beta = 20, gamma = 15)
#Add prediction line in green
lines(test$x_hat, lwd = 2, col = "green")

#Add legend
legend('topleft', col = c("red", "green"),
```

```
legend = c("Actual", "Predicted"), pch = 16)
```

---

generate_filaments *generate_filaments*

Description

A function which generates simulated SMLM datasets of filamentous objects capable of bundling. Modelled from the dynamics of plant microtubules imaged using DNA-PAINT techniques, but may be applicable to other scenarios.

Usage

```
generate_filaments(loop_number, field_settings, filament_settings,
single_bundling, bundling_dist, smoothing_settings,
cylinder_settings, optics_settings, visualise_code,
export_data, verbose)
```

Arguments

`loop_number` Number of times the code should be repeated (to be used in conjunction with export_data to generate multiple .csv datasets)

`field_settings` A vector containing three elements: the plot minimum, the plot maximum, and the number of divisions between field values.

`filament_settings` A vector containing four elements: the number of filaments, the ratio of horizontal (0) to vertical (1) filaments, the standard deviation from beginning x/y location to endpoint x/y location, and the angle below which crossing filaments will bundle.

`single_bundling` A TRUE/FALSE statement that prevents bundled filaments from going through further rounds of bundling (if FALSE, might cause unexpected results!)

`bundling_dist` A vector containing two elements: the distance between which two bundled filaments will be separated, and the number of points before crossover that should be made to be similarly parallel.

`smoothing_settings` A vector containing three elements: the probability of a point in a filament being randomised away from initial position, the radius about the initial position that randomised points may move within, and the number of rounds of Laplacian smoothing.

`cylinder_settings` A vector containing three elements: the number of points to be distributed in a cylinder about the generated splines (per filament), the width of the cylinder, and the "falloff" value (if 1 or greater, then the n-photon value is higher at the edges of the cylinder).

`optics_settings`
: A vector containing three elements: a multiplier for whole-field n-photon (gain), the randomisation of the n-photon value of each point in the field, and the optical resampling radius of each point in the field (optical error).

`visualise_code`
: A vector containing three elements: a TRUE/FALSE statement that dictates whether the simulation process should be plotted in real-time, a TRUE/FALSE statement that dictates whether a set of debug graphs should be exported to the active working directory, and the number of seconds that should be waited between blocks of code to allow the user to view the real-time graphs (set to 0 if visualise_code[1] is FALSE to avoid unnecessarily long wait times)

`export_data`
: A vector containing two elements: a TRUE/FALSE statement that dictates whether, on each iteration of loop_number, the dataset should be saved to the working directory as a .csv file, and the file prefix if export_data[1] is TRUE.

`verbose`
: A TRUE/FALSE statement that dictates whether the current place in the function loop should be printed to the terminal.

Value

A dataframe with 4 columns: the ID of each datapoint (which filament it belongs to), the x-position, the y-position, and the n-photon (simulated photonic emission), which is dependent upon the location, being boosted at the edges of the filament.

Author(s)

Andrew Buist

Examples

```
#Generate dataset
data = generate_filaments(loop_number = 1,
                          field_settings = c(0, 100, 1),
                          filament_settings = c(5, 0.3, 85, 17),
                          single_bundling = TRUE,
                          bundling_dist = c(3,10),
                          smoothing_settings = c(0.3, 5, 25),
                          cylinder_settings = c(500, 2, 1),
                          optics_settings = c(1, 0.3, 0.5),
                          visualise_code = c(FALSE, FALSE, 0),
                          export_data = c(FALSE, "data_"),
                          verbose = TRUE)

#Plot dataset coloured by ID, and opacity = n-photon
library(scales)
plot(data[,2:3], col = alpha(data[,1], alpha = data[,4]), pch = 16, cex = 0.3)
```

Index