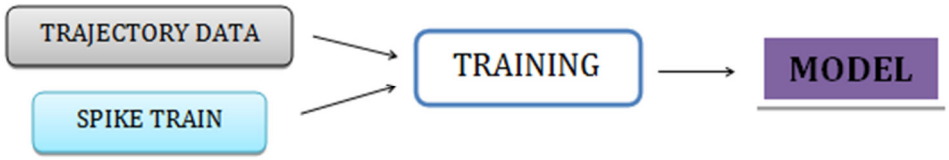
ReconTool– a machine learning toolbox for decoding neural spike trains

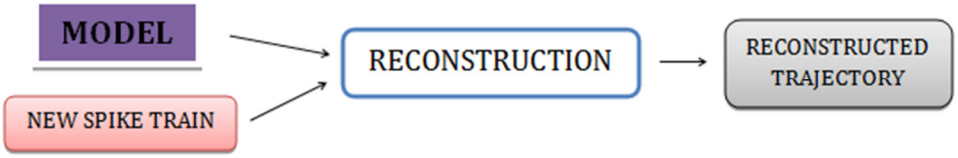
Overview

ReconTool (RECONstruction TOOLbox) is a software toolbox developed for MATLAB that uses machine learning techniques to learn from neural spiking activity and positional data of an animal, to reconstruct trajectories on novel spike trains. The core algorithm uses Bayesian inference to reconstruct the location of the animal, as described by *Zhang et al. (1998)*[1]. The toolbox is developed in such a way as to easily allow other algorithms to be developed and plugged in easily. There are two main methods, the **training** method and the **reconstruction** method.

Training Phase



Reconstruction Phase



1 Data formats

Trajectory

The trajectory data is given in the form of a matrix of size Tx3, where T is the number of sampled timestamps. The first column gives the timestamp, and the second and third columns denote the X and Y co-ordinate of the animal at that timestamp respectively. The timestamps must be provided in units of seconds.

The following image shows the start of a sample trajectory object.

	1	2	3
1	3.1324e+03	81	24
2	3.1324e+03	93	23
3	3.1324e+03	93	24
4	3.1325e+03	93	24
5	3.1325e+03	93	24
6	3.1325e+03	93	24
7	3.1326e+03	93	24
8	3.1326e+03	93	24
9	3.1326e+03	93	24
10	3.1327e+03	93	24
11	3.1327e+03	93	24

[1]K. Zhang, et al. (1998). 'Interpreting neuronal population activity by re-construction: unified framework with application to hippocampal place cells.'. Journal of neurophysiology 79(2):1017–44.

Spike train

Spiking data is provided in the form of a cell array, with N vectors, where N is the number of neurons. Each vector contains 'n' spikes, where each spike is denoted by the time at which it occurs. As with the trajectories, each spike's timestamp is given in seconds.

The following image shows the contents of a single neuron matrix in a spike train cell array.

	1	2	3
1	3.1325e+03		
2	3.1334e+03		
3	3.1352e+03		
4	3.1370e+03		
5	3.1377e+03		
6	3.1390e+03		
7	3.1391e+03		
8	3.1442e+03		
9	3.1472e+03		

Tutorial: **LOADING DATA**

A sample data set containing both position data and spike trains is provided in the ./data directory. Use the MATLAB **load** command to open it.

The '**pos**' matrix contains the trajectory of the animal, conforming to the above data specification. The spike trains are provided in the '**hpc**' cell array.

```
load lovejoy_data.mat
```

NOTE: *If the data is recorded not in units of seconds, please convert them to seconds them before proceeding to use the Toolbox. The* ***convert_to_seconds*** *function can be used to convert the given spike train and trajectory to units of seconds.*

2 Training & Reconstruction

Note: The minimum parameters required by each of the functions described below are in displayed in red, while the optional parameters are in black.

TRAINING

function [parameter_model] = training(trajectory, spikes, grid_size, intervals)

This function takes the trajectory and spikes as inputs, and grid size and training intervals can be optionally specified. The discretization size of the map is specified by a vector **[m,n]**, where the **m** and **n** are the number of row and columns respectively. The intervals are specified by a matrix of size Ix2, where I is the number of training intervals. The first column denotes the start time of the interval, and the second column denotes the end of the interval. By default, the grid size is set as 32x32, and the training is carried out over the entire given trajectory, if no specific interval is specified. The parameter model is returned, which contains firing rates, velocities, and other information calculated from the training data.

Tutorial: **TRAINING A MODEL**

Suppose we would like to train a model between seconds 3150 and 4850 of the given position and neural spiking data on a 32x32 grid.

```
param_model = training (pos, hpc, [32,32], [3150, 4850] );
```

RECONSTRUCTION

function [reconstructed_trajectory , probability_array] = reconstruction(spikes, parameter_model, intervals, time_window, compression_factor, timeunits, velocity_K)

The reconstruction method requires at least two parameters, the new spike train to be reconstructed, and the parameter model that was generated as a result of the training phase. As before, the spike train units must be seconds. Optional parameters are explained below:

- *intervals* – Reconstruction is only carried out within these intervals, if specified. It is a matrix of size Ix2, where I is the number of intervals. The first column and the second column indicate the start and the end timestamps for the reconstruction. By default, the reconstruction is carried out on one single interval spanning the whole set.
- *time_window* – Specifies the size of the time window, within which the spikes are counted during reconstruction. This parameter is algorithm dependant. By default, it is set to 1 second.
- *compression_factor* – This parameter allows for reconstruction of sequences occurring at faster timescales than the behaviour of the animal. By default, it is set to 1X.
- *velocity_K* – This constant denotes how much the reconstruction is constraint by the position of the animal in the previous time step (Please refer equation[1] 3.8) Tweaking this value *may* improve reconstruction accuracy. For This value can lie anywhere between 100 to 5000, and is set to 400 by default, if not specified.

The reconstructed output 'reconstructed_trajectory' is in the form of a cell array, containing I matrices, depending on the number of reconstruction intervals specified. Each matrix consists of Tx3 entries, where T is the number of timestamps within that interval. The second and third column corresponds to the reconstructed grid X and Y co-ordinate at that timestamp.

The second output 'probability_array' is also a cell array containing I cells, where I is the number of reconstruction intervals specified. Each cell array here contains T matrices of the training map size (default 32x32), representing the probability distribution of the expected location of the animal at each time step.

Tutorial: **RECONSTRUCTING A SPIKE TRAIN**

To reconstruct on the **hpc** spike train between 3500-3700 seconds, the following command is executed.

[traj, prob] = reconstruction (hpc, param_model, [3500, 3550]);

[1]Kumarappan, N. (2012). 'Bayesian inference of animal position from neural activity'. Master's thesis, University of Bristol.

3 Visualization Functions

There are several functions provided to visualize the data, both before and after reconstruction. This section explains the functions and how to use them.

FIRING RATES

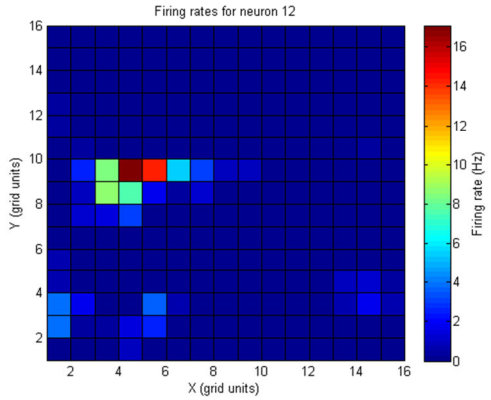
function [ ] = display_firingrate(parameter_model ,neuron_number ,checker)

This function gives a graphical display of the firing rates of a neuron, given the model after training. The firing rates are calculated and displayed in units of Hertz. Optionally, the checker grid can be disabled for a simple display by passing the third 'checker' parameter as 0.

Tutorial: **VISUALIZING FIRING RATES**
To visualize the firing rate of neuron #12, the following command is used.

```
display_firingrate ( param_model, 12 );
```

The following image shows the firing rate of a sample neuron (neuron 12 from the sample dataset).



OCCUPANCY

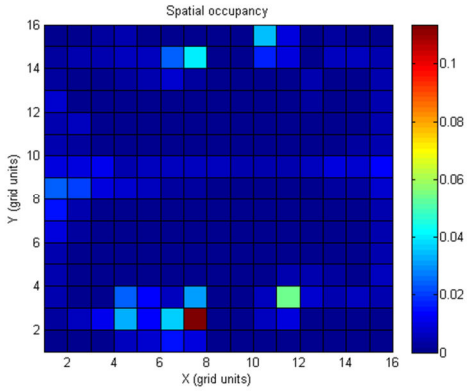
function [ ] = display_occupancy(parameter_model, checker)

This function provides a graphical display of the probability distribution of where the animal is most likely to be in. This data is generated from the input data, and is used by the reconstruction algorithm. Optionally, the checker grid can be disabled for a simple display by passing the second 'checker' parameter as 0.

Tutorial: **VISUALIZING SPATIAL OCCUPANCY**
To visualize the spatial occupancy of the animal, the following command is used.

```
display_occupancy ( param_model );
```

The following image shows the spatial occupancy of an animal from the sample dataset.



NEURON SELECTIVITY

function [ ] = display_neuronselectivity(parameter_model , checker)

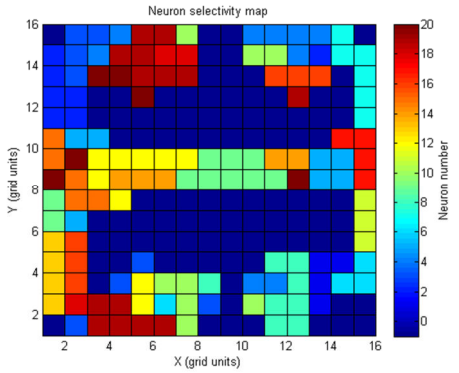
This function provides a visualization of the map, where every grid cell is given a color depending on which neuron it is most selective to. This is calculated by finding the neuron with the greatest firing rate to total firing rate ratio, for every grid cell, and denoting every neuron by a color. The resulting map gives a picture of the place fields across the maze. Optionally, the checker grid can be disabled for a simple display by passing the third 'checker' parameter as 0.

Tutorial: **VISUALIZING NEURON SELECTIVITY**

To visualize the neuron selectivity for the data used in the above examples, the following command is used.

```
display_neuronselectivity ( param_model );
```

The following plot shows the neuron selectivity visualization for the sample data.



DISPLAY PLOTS

function [ ] = display_plots(reconstruction_error , trueplot)

NOTE: This function is only useful after reconstruction, and ONLY after the *recon_error* method has been called, as it requires the *reconstruction_error* object.

This function plots the actual trajectory and overlays the reconstructed trajectory on it. For the sake of simplicity, the two coordinates of location are visualized in two individual plots. The Y axis of the plot contains either the X-coordinate or Y-

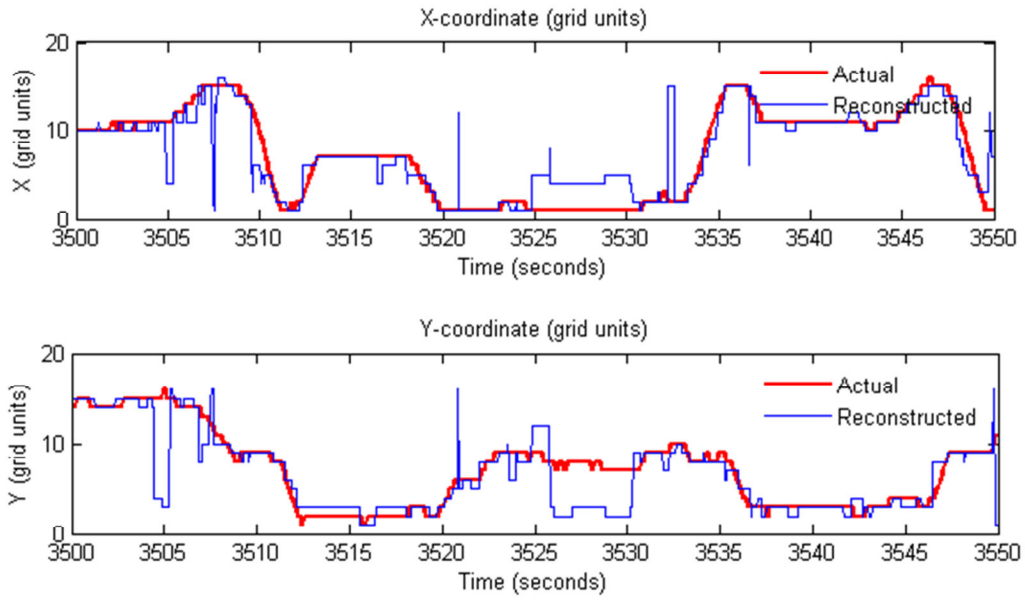
coordinate of the trajectories, and the X axis of the figure denotes *time.* If the optional *trueplot* parameter is set as 1, the plot is drawn using the default units used during training. When *trueplot* is enabled, the exact positions of the reconstructed trajectory are estimated at the centre of each reconstructed grid cell. By default, *trueplot* is set as 0 and the grid cell addresses are used for plotting. By default, *trueplot* = 0. The RED line represents the actual trajectory, while the thinner BLUE represents the reconstructed trajectory.

Tutorial: **VISUALIZING TRAJECTORIES, example I**

To plot trajectories of the previously reconstructed interval in **grid cell units**, the following command is used.

display_plots (reconstruction_error);

The following image shows the plot of X-coordinate alone on a small interval, on grid cell units.

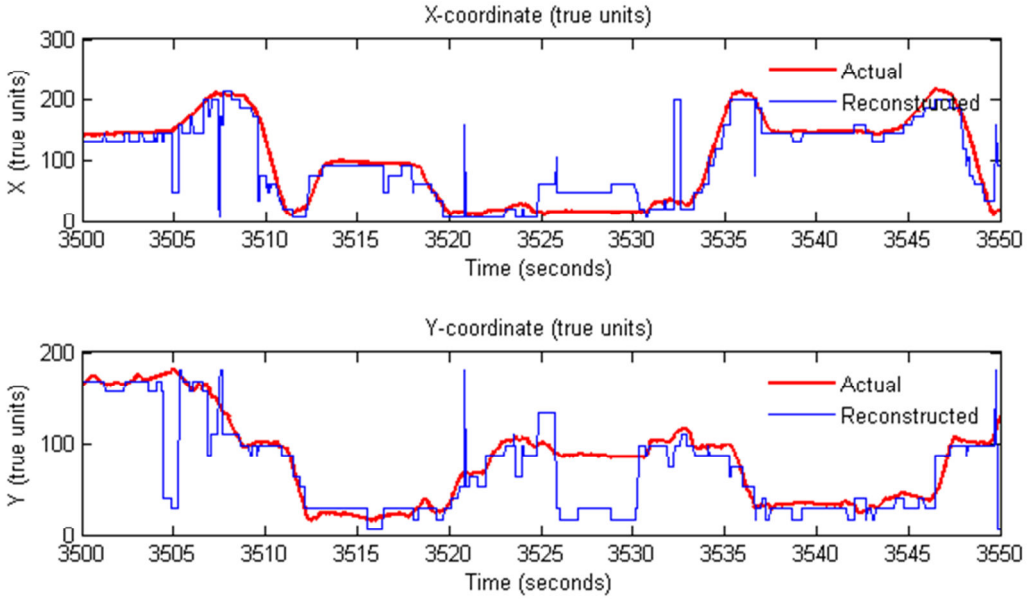


Tutorial: **VISUALIZING TRAJECTORIES, example II**

To plot both the trajectories of the previously reconstructed interval in **true units**, the following command is used.

display_plots (reconstruction_error, 1);

The following image shows the plot of both X and Y-coordinate on a small interval, in true units.



DISPLAY TRAJECTORY ANIMATION

function [ ] = display_trajectory(parameter_model, probability_array, reconstruction_error, time_per_frame)

NOTE: This function is only useful after reconstruction, and only after the *recon_error* method has been called, as it requires the *reconstruction_error* object.

This function provides a visualization of the trajectory of the animal and the probability of the reconstructed estimate at each timestep of the interval. The function takes the parameter model, probability output that is generated by the reconstruction function, and the reconstruction error output. The actual position of the animal is displayed in maroon, and the arena is displayed in blue. The probability distribution of the estimate of the animal is overlaid over this graphic at every timestep. Due to the nature of the reconstruction, the output probabilities tend to be very sharp, resulting in very few visible grid cells at any timestep. The optional time_per_frame denotes how long a frame/timestamp is displayed (in seconds), and directly affects the speed of the animation. By default, *time_per_frame* = 0.001 (seconds).

Tutorial: **ANIMATING TRAJECTORIES**
To view an animation of the reconstructed trajectory, the following command is used.

display_trajectory (param_model, prob, reconstruction_error);

4 Analysis of Reconstruction Quality

RECONSTRUCTION ERROR

function [reconstruction_error] = recon_error(trajectory, reconstructed_trajectory, model_params)

Given the initial trajectory, reconstructed trajectory and the parameter model, the function outputs a cell array, containing I matrices, where I is the number of reconstruction intervals. Each matrix is of size Tx11, where T is the number of time

steps within that interval. The resulting object contains all the information necessary to visualize and analyse the reconstruction accuracy. The eleven columns, in sequence, denote:

1. Time stamp
2. Actual grid square X-coordinate
3. Actual grid square Y-coordinate
4. Reconstructed grid square X-Coordinate
5. Reconstructed grid square Y-Coordinate
6. Square of Euclidean distance/error between real and estimated location (grid units)
7. Actual X-coordinate (in default units)
8. Actual Y-coordinate (in default units)
9. Reconstructed X-Coordinate (in estimated default units)
10. Reconstructed Y-coordinate (in estimated default units)
11. Square of Euclidean distance/error between real and estimated location (true units)

Tutorial: **CALCULATING RECONSTRUCTION ERROR**
To calculate the reconstruction error of a newly reconstructed trajectory, the following command is used.

reconstruction_error = recon_error (pos, traj, param_model);

COEFFICIENT OF DETERMINATION, R^2

function [r2] = model_R2(param_model , reconstruction_error)

This function returns a single value, which denotes the goodness of fit for that specific model, given the parameter model and the reconstruction error data. To calculate the fitness of a model, a base null model is used to compare the actual model with. The null model is a lazy model that simply reconstructs a single constant location throughout any reconstruction interval, corresponding to the grid cell that the animal is most likely to visit. The sum of squares of the Euclidian distance error is calculated and compared for both models using the following formula.

$$R^2 = 1 - \frac{Sum\ of\ squares_{model}}{Sum\ of\ squares_{null}}$$

The resulting R^2 value gives the goodness of fit of the model for that data, by comparison with the performance across the same data by the null model. A value of 1 is ideal, and indicates that the model has zero-error and is perfect. A value of zero means that the model performs just as good/bad as the null model. A negative value indicates that the model performs worse than the null model on the given data. This measure of evaluating model quality is superior as it is independent of parameters such as map discretization size, since it compares with a base model across the same data.

Tutorial: **CALCULATING GOODNESS OF FIT**
To calculate the goodness of fit of a model over a reconstructed interval, the following command is used:

R2 = model_R2 (param_model , reconstruction_error);

CROSS VALIDATION

function [results] = crossvalidation(trajectory, spikes, interval, folds, compression_factor, time_window, velocity_K)

This function automatically performs cross validation tests on the given spiking train and positional data, and returns the results for every fold. The performance of the model on a fold is determined by the model's goodness of fit R^2 value, which is calculated by the *model_R2* function. The output results object is a matrix of size Fx3, where F is the number of folds, and the first and second column correspond to the start and the end time of the fold. The third column gives the R^2 value of the model on that fold. The number of folds can be specified as an optional parameter, but by default a 10 fold CV is used. All the other parameters behave the same way as in the reconstruction method, and share the same defaults.

Tutorial: **PERFORMING A CROSS-VALIDATION**
Suppose we wish to a perform a 10-fold CV in the interval between 3500 – 4200 seconds. The following command is used.

```
scores_cv = crossvalidation ( pos , hpc, [3500,4200], 10 );
```

4 Miscellaneous Functions

SHIFT ORIGIN

function [fixed_trajectory] = shift_origin(trajectory)

This function returns a fixed trajectory which is normalized to zero. This can be useful in cases when the recorded data is not normalized properly.

CONVERT TO SECONDS

function [pos, spikes] = convert_to_seconds(pos, spikes, n)

This function returns the provided position and spike trains in proper units of seconds. n indicates that the units for the data provided are recorded in 1/n seconds. Eg. To convert data recorded in milliseconds (1/1000th of a second), n is set as 1000. By default, n=1000.