OpenCortex Documentation

Release 0.1.11

OpenCortex authors and contributors

Oct 19, 2021

Contents

CHAPTER 1

Introduction

In progress. . .

CHAPTER 2

Installation

2.1 Requirements

Work in progress. . .

CHAPTER 3

Developer documentation

3.1 How to contribute

Contact: p dot gleeson at gmail dot com

3.2 opencortex

3.2.1 opencortex Package

See *opencortex.core* for the main module for user interaction with OpenCortex.

opencortex Package

opencortex.__init__.**print_comment**(*text*, *print_it=False*)
Print a comment only if print_it == True

opencortex.__init__.**print_comment_v**(*text*)
Always print the comment

opencortex.__init__.**set_verbose**(*value=True*)

Subpackages

opencortex core Package

This is the main module for user interaction with OpenCortex.

opencortex.core Package

opencortex.core.**add_exp_one_syn**(*nml_doc*, *id*, *gbase*, *erev*, *tau_decay*)
Adds an <expOneSynapse> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/Synapses.html#expOneSynapse

Returns the class created.

opencortex.core.**add_exp_two_syn**(*nml_doc*, *id*, *gbase*, *erev*, *tau_rise*, *tau_decay*)
Adds an <expTwoSynapse> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/Synapses.html#expTwoSynapse

Returns the class created.

opencortex.core.**add_gap_junction_synapse**(*nml_doc*, *id*, *conductance*)
Adds a <gapJunction> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/Synapses.html#gapJunction

Returns the class created.

opencortex.core.**add_inputs_to_population**(*net*, *id*, *population*, *input_comp_id*, *number_per_cell=1*, *all_cells=False*, *only_cells=None*, *segment_ids=[0]*, *fraction_alongs=[0.5]*, *weights=1*)
Add current input to the specified population. Attributes:

net reference to the network object previously created

id id of the <inputList> to be created

population the <population> to be targeted

input_comp_id id of the component to be used for the input (e.g. added with add_pulse_generator())

number_per_cell how many inputs to apply to each cell of the population. Default 1

all_cells Whether to target all cells. Default False

only_cells Which specific cells to target. List of ids. Default None

segment_ids List of segment ids to place inputs onto on each cell. Either list of 1 value or list of number_per_cell entries. Default [0]

fraction_alongs List of fractions along the specified segments to place inputs onto on each cell. Either list of 1 value or list of number_per_cell entries. Default [0.5]

weights Either a scalar value (all weights set to this for all inputs), or a function to pick a (random) value per input. Default 1

opencortex.core.**add_poisson_firing_synapse**(*nml_doc*, *id*, *average_rate*, *synapse_id*)
Adds a <poissonFiringSynapse> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/Inputs.html#poissonFiringSynapse

Returns the class created.

opencortex.core.**add_population_in_rectangular_region**(*net*, *pop_id*, *cell_id*, *size*, *x_min*, *y_min*, *z_min*, *x_size*, *y_size*, *z_size*, *cell_bodies_overlap=True*, *store_soma=False*, *population_dictionary=None*, *cell_diameter_dict=None*, *color=None*)

Method which creates a cell population in the NeuroML2 network and distributes these cells in the rectangular region. Input arguments are:

net reference to the network object previously created

pop_id population id

cell_id cell component id

size size of a population

x_min lower x bound of a rectangular region

y_min lower y bound of a rectangular region

z_min lower z bound of a rectangular region

x_size width of a rectangular region along x axis

y_size width of a rectangular region along y axis

z_size width of a rectangular region along z axis

cell_bodies_overlap boolean value which defines whether cell somata can overlap; default is set to True

store_soma boolean value which specifies whether soma positions have to be returned in the output array; default is set to False

population_dictionary optional argument in the format returned by opencortex.utils.add_populations_in_rectangular_layers; default value is None but it must be specified when cell_bodies_overlap is set to False

cell_diameter_dict optional argument in the format {'cell_id1': soma diameter of type 'float', 'cell_id2': soma diameter of type 'float'}; default is None but it must be specified when cell_bodies_overlap is set to False

color optional color (which will be put through to annotation in generated NeuroML); RGB format, 3 floats 0->1, e.g. 1 0 0 for red; default is None

opencortex.core.**add_probabilistic_projection**(*net*, *prefix*, *presynaptic_population*, *postsynaptic_population*, *synapse_id*, *connection_probability*, *delay=0*, *weight=1*)

Add a projection between *presynaptic_population* and *postsynaptic_population* with probability of connection between each pre & post pair of cells given by *connection_probability*. Attributes:

net reference to the network object previously created

prefix prefix to use in the id of the projection

presynaptic_population presynaptic population e.g. added via add_population_in_rectangular_region()

postsynaptic_population postsynaptic population e.g. added via add_population_in_rectangular_region()

synapse_id id of synapse previously added, e.g. added with add_exp_two_syn()

connection_probability For each pre syn cell i and post syn cell j, where i!=j, the chance they will be connected is given by this

delay optional delay for each connection, default 0 ms

weight optional weight for each connection, default 1

opencortex.core.**add_pulse_generator**(*nml_doc*, *id*, *delay*, *duration*, *amplitude*)

Adds a <pulseGenerator> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/Inputs.html#pulseGenerator

Returns the class created.

opencortex.core.**add_single_cell_population**(*net*, *pop_id*, *cell_id*, *x=0*, *y=0*, *z=0*, *color=None*)

Add a population with id *pop_id* containing a single instance of cell *cell_id*. Optionally specify (*x*,'y','z') and the population *color*.

opencortex.core.**add_spike_source_poisson**(*nml_doc*, *id*, *start*, *duration*, *rate*)

Adds a <SpikeSourcePoisson> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/PyNN.html#SpikeSourcePoisson

Returns the class created.

opencortex.core.**add_targeted_electrical_projection**(*nml_doc*, *net*, *prefix*, *presynaptic_population*, *postsynaptic_population*, *targeting_mode*, *synapse_list*, *number_conns_per_cell*, *pre_segment_group*, *post_segment_group*)

Adds (electrical, gap junction mediated) projection from *presynaptic_population* to *postsynaptic_population*, specifically limiting connections presynaptically to *pre_segment_group* and postsynaptically to *post_segment_group*.

opencortex.core.**add_targeted_inputs_to_population**(*net*, *id*, *population*, *input_comp_id*, *segment_group*, *number_per_cell=1*, *all_cells=False*, *only_cells=None*, *weights=1*)

Add current input to the specified population. Attributes:

net reference to the network object previously created

id id of the <inputList> to be created

population the <population> to be targeted

input_comp_id id of the component to be used for the input (e.g. added with add_pulse_generator())

segment_group which segment group on the target cells to limit input locations to

number_per_cell How many inputs to apply to each cell of the population. Default 1

all_cells whether to target all cells. Default False

only_cells which specific cells to target. List of ids. Default None

weights Either a scalar value (all weights set to this for all inputs), or a function to pick a (random) value per input. Default 1

opencortex.core.**add_targeted_projection**(*net*, *prefix*, *presynaptic_population*, *postsynaptic_population*, *targeting_mode*, *synapse_list*, *number_conns_per_cell*, *pre_segment_group=None*, *post_segment_group=None*, *delays_dict=None*, *weights_dict=None*)

Adds (chemical, event based) projection from *presynaptic_population* to *postsynaptic_population*, specifically limiting connections presynaptically to *pre_segment_group* and postsynaptically to *post_segment_group*.

net reference to the network object previously created

prefix prefix to use in the id of the projection

presynaptic_population presynaptic population e.g. added via add_population_in_rectangular_region()

postsynaptic_population postsynaptic population e.g. added via add_population_in_rectangular_region()

targeting_mode a string that specifies the targeting mode: 'convergent' or 'divergent'

synapse_list the list of synapse ids that correspond to the individual receptor components on the physical synapse, e.g. the first element is the id of the AMPA synapse and the second element is the id of the NMDA synapse; these synapse components will be mapped onto the same location of the target segment

number_conns_per_cell number of connections to make on each cell in the postsynaptic population (when targeting_mode='convergent') or from each cell in the presynaptic population (when targeting_mode='divergent')

pre_segment_group which segment_group to target connennections from on the presynaptic population, e.g. axon_group. This can be left out or set to None if the presynaptic component has no morphology

post_segment_group which segment_group to target connennections from on the postsynaptic population, e.g. dendrite_group. This can be left out or set to None if the postsynaptic component has no morphology

delays_dict optional dictionary that specifies the delays (in ms) for individual synapse components, e.g. {'NMDA':5.0} or {'AMPA':3.0,'NMDA':5}

weights_dict optional dictionary that specifies the weights for individual synapse components, e.g. {'NMDA':1} or {'NMDA':1,'AMPA':2}

opencortex.core.**add_transient_poisson_firing_synapse**(*nml_doc*, *id*, *average_rate*, *delay*, *duration*, *synapse_id*)

Adds a <transientPoissonFiringSynapse> element to the document. See the definition of the behaviour of this here: https://www.neuroml.org/NeuroML2CoreTypes/Inputs.html#transientPoissonFiringSynapse

Returns the class created.

opencortex.core.**add_voltage_clamp_triple**(*nml_doc*, *id*, *delay*, *duration*, *conditioning_voltage*, *testing_voltage*, *return_voltage*, *simple_series_resistance*, *active='1'*)

Adds a <voltageClampTriple> element to the document. See the definition of the behaviour of this here: https://github.com/NeuroML/NeuroML2/blob/development/Schemas/NeuroML2/NeuroML_v2beta5.xsd

Returns the class created.

opencortex.core.**generate_lems_simulation**(*nml_doc*, *network*, *nml_file_name*, *duration*, *dt*, *target_dir='.'*, *include_extra_lems_files=[]*, *gen_plots_for_all_v=True*, *plot_all_segments=False*, *gen_plots_for_quantities={}*, *gen_plots_for_only_populations=[]*, *gen_saves_for_all_v=True*, *save_all_segments=False*, *gen_saves_for_only_populations=[]*, *gen_saves_for_quantities={}*, *gen_spike_saves_for_all_somas=False*, *gen_spike_saves_for_only_populations=[]*, *gen_spike_saves_for_cells={}*, *spike_time_format='ID_TIME'*, *report_file_name=None*, *lems_file_name=None*, *lems_file_generate_seed=12345*, *simulation_seed=12345*, *verbose=False*)

Generate a LEMS simulation file with which to run simulations of the network. Generated LEMS files can be run with jNeuroML (or converted to simulator specific formats, e.g. NEURON, and run)

opencortex.core.**generate_network**(*reference*, *network_seed=1234*, *temperature='32degC'*)

Generate a network which will contain populations, projections, etc. Arguments:

reference the reference to use as the id for the network

network_seed optional, will be used for random elements of the network, e.g. placement of cells in 3D

temperature optional, will be specified in network and used in temperature dependent elements, e.g. ion channels with Q10. Default: 32degC

opencortex.core.**include_cell_prototype**(*nml_doc*, *cell_nml2_path*)
Add a NeuroML2 file containing a cell definition

opencortex.core.**include_neuroml2_cell**(*nml_doc*, *cell_nml2_path*, *cell_id*, *channels_also=True*)
Add a cell with id *cell_id* which is in *cell_nml2_path* to the build document, along with all its channels (if channels_also==True)

opencortex.core.**include_neuroml2_cell_and_channels**(*nml_doc*, *cell_nml2_path*, *cell_id*)
DEPRECATED TODO: remove, due to include_neuroml2_cell Add a cell with id *cell_id* which is in *cell_nml2_path* to the build document, along with all its channels

opencortex.core.**include_neuroml2_file**(*nml_doc*, *nml2_file_path*)
Add a NeuroML2 file containing definitions of elements which can be used in the network

opencortex.core.**include_opencortex_cell**(*nml_doc*, *reference*)
Include a cell from the standard set of NeuroML2 cells included with OpenCortex. See https://github.com/OpenSourceBrain/OpenCortex/tree/master/NeuroML2/prototypes.

opencortex.core.**save_network**(*nml_doc*, *nml_file_name*, *validate=True*, *format='xml'*, *max_memory=None*, *target_dir='./'*, *use_subfolder=True*)
Save the contents of the built NeuroML document, including the network to the file specified by *nml_file_name*, optionally specifying the *target_dir*

opencortex.core.**simulate_network**(*lems_file_name*, *simulator*, *max_memory='400M'*, *nogui=True*, *load_saved_data=False*, *reload_events=False*, *plot=False*, *verbose=True*, *num_processors=1*)
Run a simulation of the LEMS file *lems_file_name* using target platform *simulator*

opencortex build Package

Utility functions for OpenCortex. See *opencortex.core* for the main module for user interaction with OpenCortex.

opencortex.build Package

opencortex.build.**add_advanced_inputs_to_population**(*net*, *id*, *population*, *input_id_list*, *seg_length_dict*, *subset_dict*, *universal_target_segment*, *universal_fraction_along*, *all_cells=False*, *only_cells=None*, *weight_dict=None*)
Do not use. Subject to change!!!

This method distributes the poisson input synapses on the specific segment groups of target cells. Input arguments to this method:

net- libNeuroML network object;

id - unique string that tags the input group created by the method;

population - libNeuroML population object of a target population;

input_id_list - this is a list that stores lists of poisson synapse ids or pulse generator ids; if len(input_id_list)== (num of target cells) then each target cell, specified by only_cells or all_cells, has a unique list input components;

if len(input_id_list != num, then add_advanced_inputs_to_population assumes that all cells share the same list of input components and thus uses input_id_list[0]. Note that all of the input components (e.g. differing in delays) per given list of input components are mapped on the same membrane point on the target segment of a given cell.

seg_length_dict - a dictionary whose keys are the ids of target segment groups and the values are the segment length dictionaries in the format returned by make_target_dict();

subset_dict - a dictionary whose keys are the ids of target segment groups and the corresponding dictionary values define the desired number of synaptic connections per target segment group per each postsynaptic cell;

universal_target_segment - this should be set to None if subset_dict and seg_length_dict are used; alternatively, universal_target_segment specifies a single target segment on all of the target cells for all input components; then seg_length_dict and subset_dict must be set to None.

universal_fraction_along - this should be set to None if subset_dict and seg_length_dict are used; alternatively, universal_target_fraction specifies a single value of fraction along on all of the target segments for all target cells and all input components; then seg_length_dict and subset_dict must bet set to None;

all_cells - default value is set to False; if all_cells==True then all cells in a given population will receive the inputs;

only_cells - optional variable which stores the list of ids of specific target cells; cannot be set together with all_cells.

weight_dict - id of cell vs weight for each connection

opencortex.build.**add_chem_spatial_projection**(*net*, *proj_array*, *presynaptic_population*, *postsynaptic_population*, *targeting_mode*, *synapse_list*, *pre_seg_target_dict*, *post_seg_target_dict*, *subset_dict*, *distance_rule*, *pre_cell_positions*, *post_cell_positions*, *delays_dict*, *weights_dict*)

This method adds the divergent distance-dependent chemical projection. The input arguments are as follows:

net - the network object created using libNeuroML API (neuroml.Network());

proj_array - list which stores the projections of class neuroml.Projection; each projection has unique synapse component (e.g. AMPA , NMDA or GABA); thus for each projection the list position in the proj_array must be identical to the list position of the corresponding synapse id in the synapse_list;

presynaptic_population - object corresponding to the presynaptic population in the network;

postsynaptic_population - object corresponding to the postsynaptic population in the network;

targeting_mode - a string that specifies the targeting mode: 'convergent' or 'divergent';

synapse_list - the list of synapse ids that correspond to the individual receptor components on the physical synapse, e.g. the first element is the id of the AMPA synapse and the second element is the id of the NMDA synapse; these synapse components will be mapped onto the same location of the target segment;

pre_seg_target_dict - a dictionary whose keys are the ids of presynaptic segment groups and the values are dictionaries in the format returned by make_target_dict();

post_seg_target_dict - a dictionary whose keys are the ids of target segment groups and the values are dictionaries in the format returned by make_target_dict();

subset_dict - a dictionary whose keys are the ids of target segment groups; interpretation of the corresponding dictionary values depends on the targeting mode:

Case I, targeting mode = 'divergent' - the desired number of synaptic connections made by each presynaptic cell per given target segment group of postsynaptic cells;

Case II, targeting mode = 'convergent' - the desired number of synaptic connections per target segment group per each postsynaptic cell;

alternatively, subset_dict can be a number that specifies the total number of synaptic connections (either divergent or convergent) irrespective of target segment groups.

Note: the chemical connection is made only if distance-dependent probability is higher than some random number random.random(); thus, the actual numbers of connections made

according to the distance-dependent rule might be smaller than the numbers of connections specified by subset_dict; subset_dict defines the upper bound for the

number of connections.

distance_rule - string which defines the distance dependent rule of connectivity - soma to soma distance must be represented by the string character 'r';

pre_cell_positions- array specifying the cell positions for the presynaptic population; the format is an array of [x coordinate, y coordinate, z coordinate];

post_cell_positions- array specifying the cell positions for the postsynaptic population; the format is an array of [x coordinate, y coordinate, z coordinate];

delays_dict - optional dictionary that specifies the delays (in ms) for individual synapse components, e.g. {'NMDA':5.0} or {'AMPA':3.0,'NMDA':5};

weights_dict - optional dictionary that specifies the weights for individual synapse components, e.g. {'NMDA':1} or {'NMDA':1,'AMPA':2}.

opencortex.build.**add_elect_connection**(*projection*, *id*, *presynaptic_population*, *pre_cell_id*, *pre_seg_id*, *postsynaptic_population*, *post_cell_id*, *post_seg_id*, *gap_junction_id*, *pre_fraction=0.5*, *post_fraction=0.5*)

Add a single electrical connection (via a gap junction) to a projection between *presynaptic_population* and *postsynaptic_population*

opencortex.build.**add_elect_spatial_projection**(*net*, *proj_array*, *presynaptic_population*, *postsynaptic_population*, *targeting_mode*, *synapse_list*, *pre_seg_target_dict*, *post_seg_target_dict*, *subset_dict*, *distance_rule*, *pre_cell_positions*, *post_cell_positions*)

This method adds the divergent or convergent electrical projection depending on the input argument targeting_mode. The input arguments are as follows:

net - the network object created using libNeuroML API (neuroml.Network());

proj_array - dictionary which stores the projections of class neuroml.ElectricalProjection; each projection has unique gap junction component; thus for each projection the list position in the proj_array must be identical to the list position of the corresponding gap junction id in the synapse_list;

presynaptic_population - object corresponding to the presynaptic population in the network;

postsynaptic_population - object corresponding to the postsynaptic population in the network;

targeting_mode - a string that specifies the targeting mode: 'convergent' or 'divergent';

synapse_list - the list of gap junction (synapse) ids that correspond to the individual gap junction components on the physical contact; these components will be mapped onto the same location of the target segment;

pre_seg_target_dict - a dictionary whose keys are the ids of presynaptic segment groups and the values are dictionaries in the format returned by make_target_dict();

post_seg_target_dict - a dictionary whose keys are the ids of target segment groups and the values are dictionaries in the format returned by make_target_dict();

subset_dict - a dictionary whose keys are the ids of target segment groups; interpretation of the corresponding dictionary values depends on the targeting mode:

Case I, targeting mode = 'divergent' - the number of synaptic connections made by each presynaptic cell per given target segment group of postsynaptic cells;

Case II, targeting mode = 'convergent' - the number of synaptic connections per target segment group per each postsynaptic cell;

alternatively, subset_dict can be a number that specifies the total number of synaptic connections (either divergent or convergent) irrespective of target segment groups.

Note: the electrical connection is made only if distance-dependent probability is higher than some random number random.random(); thus, the actual numbers of connections made

according to the distance-dependent rule might be smaller than the numbers of connections specified by subset_dict; subset_dict defines the upper bound for the

number of connections.

distance_rule - string which defines the distance dependent rule of connectivity - soma to soma distance must be represented by the string character 'r';

pre_cell_positions- array specifying the cell positions for the presynaptic population; the format is an array of [x coordinate, y coordinate, z coordinate];

post_cell_positions- array specifying the cell positions for the postsynaptic population; the format is an array of [x coordinate, y coordinate, z coordinate];

opencortex.build.**add_population_in_cylindrical_region**(*net*, *pop_id*, *cell_id*, *size*, *cyl_radius*, *lower_bound_dim3*, *upper_bound_dim3*, *base_dim1='x'*, *base_dim2='z'*, *cell_bodies_overlap=True*, *store_soma=False*, *population_dictionary=None*, *cell_diameter_dict=None*, *num_of_polygon_sides=None*, *positions_of_vertices=None*, *constants_of_sides=None*, *color=None*)

Method which create a cell population in the NeuroML2 network and distributes these cells in the cylindrical region. Input arguments are as follows:

net - reference to the libNeuroML network object;

pop_id - population id;

cell_id - cell component id;

size - size of a population;

cyl_radius - radius of a cylindrical column in which cells will be distributed;

lower_bound_dim3 - lower bound of the cortical column height;

upper_bound_dim3 - upper bound of the cortical column height;

base_dim1 - specifies which of the 'x', 'y' and 'z' axis corresponds to the first dimension of the transverse plane of the cortical column;

base_dim2 - specifies which of the 'x', 'y' and 'z' axis corresponds to the second dimension of the transverse plane of the cortical column;

cell_bodies_overlap - boolean value which defines whether cell somata can overlap; default is set to True;

store_soma -boolean value which specifies whether soma positions have to be stored in the output array; default is set to False;

population_dictionary - optional argument in the format returned by add_populations_in_rectangular_layers; default value is None but it must be specified when cell_bodies_overlap is set to False;

cell_diameter_dict - optional argument in the format {'cell_id1': soma diameter of type 'float', 'cell_id2': soma diameter of type 'float'}; default is None but it must be specified when cell_bodies_overlap is set to False.

num_of_polygon_sides - optional argument which specifies the number of sides of regular polygon which is inscribed in th
is None, thus cylindrical but not polygonal shape is built.

positions_of_vertices - optional argument which specifies the list of coordinates [dim1, dim2] of vertices of a regular polygon; must be specified if num_of_polygon_sides is not None; automatic generation of this list is wrapped inside the utils method add_populations_in_cylindrical_layers();

constants_of_sides - optional argument which specifies the list of y=ax +b coefficients [a, b] which define the lines between vertices of a regular polygon; if y= b for all values then list element should specify as [None, b]; if x= b for all values of y then list element should specify as [b, None]; Note that constants_of_sides must be specified if num_of_polygon_sides is not None; automatic generation of this list is wrapped inside the utils method add_populations_in_cylindrical_layers();

color - optional color, default is None.

opencortex.build.**add_probabilistic_projection_list**(*net*, *presynaptic_population*, *postsynaptic_population*, *synapse_list*, *connection_probability*, *delay=0*, *weight=1*, *presynaptic_population_list=True*, *postsynaptic_population_list=True*, *clipped_distributions=True*, *std_delay=None*, *std_weight=None*)

Modification of the method *add_probabilistic_projection()* to allow multiple synaptic components per physical projection; specifically works for networks containing single-compartment neuronal models. This method also allows gaussian variation in synaptic weight and delay; it also accepts populations that do not necessarily have the type attribute in <population> set to *populationList* .

opencortex.build.**add_projection_based_inputs**(*net*, *id*, *population*, *input_id_list*, *weight_list*, *synapse_id*, *seg_length_dict*, *subset_dict*, *universal_target_segment*, *universal_fraction_along*, *all_cells=False*, *only_cells=None*)

This method builds input projections between the input components and target population. Input arguments to this method:

net- libNeuroML network object;

id - unique string that tags the input group created by the method;

population - libNeuroML population object of a target population;

input_id_list - this is a list that stores lists of instance ids of SpikeSourcePoisson component types;

if len(input_id_list)== (num of target cells) then each target cell, specified by only_cells or all_cells, has a unique list input components; if len(input_id_list != num, then add_advanced_inputs_to_population assumes that all cells share the same list of input components and thus uses input_id_list[0]. Note that all of the input components (e.g. differing in delays) per given list of input components are mapped on the same membrane point on the target segment of a given cell.

weight_list - lists of connection weights for the input components specified by input_id_list; it must take the same format as input_id_list;

synapse_id - unique synapse id for all input components specified in input_id_list;

seg_length_dict - a dictionary whose keys are the ids of target segment groups and the values are the segment length dictionaries in the format returned by make_target_dict();

subset_dict - a dictionary whose keys are the ids of target segment groups and the corresponding dictionary values define the desired number of synaptic connections per target segment group per each postsynaptic cell;

universal_target_segment - this should be set to None if subset_dict and seg_length_dict are used; alternatively, universal_target_segment specifies a single target segment on all of the target cells for all input components; then seg_length_dict and subset_dict must be set to None.

universal_fraction_along - this should be set to None if subset_dict and seg_length_dict are used; alternatively, universal_target_fraction specifies a single value of fraction along on all of the target segments for all target cells and all input components; then seg_length_dict and subset_dict must bet set to None;

all_cells - default value is set to False; if all_cells==True then all cells in a given population will receive the inputs;

only_cells - optional variable which stores the list of ids of specific target cells; cannot be set together with all_cells.

opencortex.build.**add_synapses**(*nml_doc*, *nml2_path*, *synapse_list*, *synapse_tag=True*)

opencortex.build.**add_targeted_projection_by_dicts**(*net*, *proj_array*, *presynaptic_population*, *postsynaptic_population*, *targeting_mode*, *synapse_list*, *pre_seg_target_dict*, *post_seg_target_dict*, *subset_dict*, *delays_dict=None*, *weights_dict=None*)

This method adds the divergent or convergent chemical projection depending on the input argument targeting_mode. The input arguments are as follows:

net - the network object created using libNeuroML API (neuroml.Network());

proj_array - list which stores the projections of class neuroml.Projection; each projection has unique synapse component (e.g. AMPA , NMDA or GABA); thus for each projection the list position in the proj_array must be identical to the list position of the corresponding synapse id in the synapse_list;

presynaptic_population - object corresponding to the presynaptic population in the network;

postsynaptic_population - object corresponding to the postsynaptic population in the network;

targeting_mode - a string that specifies the targeting mode: 'convergent' or 'divergent';

synapse_list - the list of synapse ids that correspond to the individual receptor components on the physical synapse, e.g. the first element is the id of the AMPA synapse and the second element is the id of the NMDA synapse; these synapse components will be mapped onto the same location of the target segment;

pre_seg_target_dict - a dictionary whose keys are the ids of presynaptic segment groups and the values are dictionaries in the format returned by make_target_dict();

post_seg_target_dict - a dictionary whose keys are the ids of target segment groups and the values are dictionaries in the format returned by make_target_dict();

subset_dict - a dictionary whose keys are the ids of target segment groups; interpretation of the corresponding dictionary values depends on the targeting mode:

Case I, targeting mode = 'divergent' - the number of synaptic connections made by each presynaptic cell per given target segment group of postsynaptic cells;

Case II, targeting mode = 'convergent' - the number of synaptic connections per target segment group per each postsynaptic cell;

alternatively, subset_dict can be a number that specifies the total number of synaptic connections (either divergent or convergent) irrespective of target segment groups.

delays_dict - optional dictionary that specifies the delays (in ms) for individual synapse components, e.g. {'NMDA':5.0} or {'AMPA':3.0,'NMDA':5};

weights_dict - optional dictionary that specifies the weights for individual synapse components, e.g. {'NMDA':1} or {'NMDA':1,'AMPA':2}.

opencortex.build.**copy_nml2_source**(*dir_to_project_nml2*, *primary_nml2_dir*, *electrical_synapse_tags*, *chemical_synapse_tags*, *extra_channel_tags=[]*)

This method copies the individual NeuroML2 model components from the primary source dir to corresponding component folders in the target dir: "synapses", "gapJunctions",

"channels" and "cells" are created in the dir_to_project_nml2.

opencortex.build.**distance**(*p*, *q*)

opencortex.build.**extract_seg_ids**(*cell_object*, *target_compartment_array*, *targeting_mode*)

This method extracts the segment ids that map on the target segment groups or individual segments. cell_object is the loaded cell object using neuroml.loaders.NeuroMLLoader, target_compartment_array is an array of target compartment names (e.g. segment group ids or individual segment names) and targeting_mode is one of the strings: "segments" or "segGroups".

opencortex.build.**find_constrained_cell_position**(*num_of_polygon_sides*, *cyl_radius*, *lower_bound_dim3*, *upper_bound_dim3*, *positions_of_vertices*, *constants_of_sides*)

Method to find a constrained position of the cell; used inside the method add_population_in_cylindrical_region().

opencortex.build.**get_pre_and_post_segment_ids**(*proj*)

This method extracts the lists of pre and post segment ids per given projection. Can be used when substituting the cell types from one NeuroML2 network to the other.

return pre_segment_ids, post_segment_ids

opencortex.build.**get_seg_lengths**(*cell_object*, *target_segments*)

This method constructs the cumulative distribution of target segments and the corresponding list of target segment ids. Input arguments: cell_object - object created using libNeuroML API which corresponds to the target cell; target_segments - the list of target segment ids.

opencortex.build.**get_soma_diameter**(*cell_name*, *cell_type=None*, *dir_to_cell=None*)

Method to obtain a diameter of a cell soma.

opencortex.build.**get_target_cells**(*population*, *fraction_to_target*, *list_of_xvectors=None*, *list_of_yvectors=None*, *list_of_zvectors=None*)

This method returns the list of target cells according to which fraction of randomly selected cells is targeted and

whether these cells are localized in the specific rectangular regions of the network. These regions are specified by list_of_xvectors, list_of_yvectors and list_of_zvectors. These lists must have the same length.

The input variable list_of_xvectors stores the lists whose elements define the left and right margins of the target rectangular regions along the x dimension.

Similarly, the input variables list_of_yvectors and list_of_zvectors store the lists whose elements define the left and right margins of the target rectangular regions along the y and z dimensions, respectively.

opencortex.build.**get_target_segments**(*seg_specifications*, *subset_dict*)

This method generates the list of target segments and target fractions per cell according to two types of input dictionaries: seg_specifications - a dictionary in the format returned by make_target_dict(); keys are target group names or individual segment names and the corresponding values are dictionaries with keys 'LengthDist' and 'SegList', as returned by the get_seg_lengths; subset_dict - a dictionary whose keys are target group names or individual segment names; each key stores the corresponding number of connections per target group.

opencortex.build.**make_target_dict**(*cell_object*, *target_segs*)

This method constructs the dictionary whose keys are the names of target segment groups or individual segments and the corresponding values are dictionaries with keys 'LengthDist' and 'SegList', as returned by the get_seg_lengths. Input arguments are as follows:

cell_object - object created using libNeuroML API which corresponds to the target cell; target_segs - a dictionary in the format returned by the method extract_seg_ids(); the keys are the ids of target segment groups or names of individual segments and the values are lists of corresponding target segment ids.

opencortex.build.**remove_component_dirs**(*dir_to_project_nml2*, *list_of_cell_ids*, *extra_channel_tags=None*)

This method removes the sufolder strings of NeuroML2 component types (if they exist) from the 'includes' of each NeuroML2 cell in the target dir.

Target directory is specified by the input argument dir_to_project_nml2.

opencortex utils Package

Utility functions for OpenCortex. See *opencortex.core* for the main module for user interaction with OpenCortex.

opencortex.utils Package

opencortex.utils.**add_populations_in_cylindrical_layers**(*net*, *boundaryDict*, *popDict*, *radiusOfCylinder*, *storeSoma=True*, *cellBodiesOverlap=True*, *cellDiameterArray=None*, *numOfSides=None*)

This method distributes the cells in cylindrical layers. The input arguments:

net - libNeuroML network object;

popDict - a dictionary whose keys are unique cell population ids; each key entry stores a tuple of five elements: population size, Layer tag, cell model id, compartmentalization, color;

layer tags (of type string) must make up the keys() of boundaryDict;

boundaryDict have layer pointers as keys; each entry stores the left and right bound of the layer in the list format , e.g. [L3_min, L3_max]

x_vector - a vector that stores the left and right bounds of the cortical column along x dimension

y_vector - a vector that stores the left and right bounds of the cortical column along y dimension

radiusOfCylinder - radius of cylindrical column in which cortical cells will be distributed.

storeSoma - specifies whether soma positions have to be stored in the output array (default is set to True);

cellBodiesOverlap - boolean value which defines whether cell somata can overlap; default is set to True;

cellDiameterArray - optional dictionary of cell model diameters required when cellBodiesOverlap is set to False;

numOfSides - optional argument which specifies the number of sides of regular polygon which is inscribed in the cylindrical column of a given radius; default value is None, thus cylindrical but not polygonal shape is built.

This method returns the dictionary; each key is a unique cell population id and the corresponding value is a dictionary which refers to libNeuroML population object (key 'PopObj') and cell position array ('Positions') which by default is None.

opencortex.utils.**add_populations_in_rectangular_layers**(*net*, *boundaryDict*, *popDict*, *x_vector*, *z_vector*, *storeSoma=True*, *cellBodiesOverlap=True*, *cellDiameterArray=None*)

This method distributes the cells in rectangular layers. The input arguments:

net - libNeuroML network object;

popDict - a dictionary whose keys are unique cell population ids; each key entry stores a tuple of five elements: population size, Layer tag, cell model id, compartmentalization, color;

layer tags (of type string) must make up the keys() of boundaryDict;

boundaryDict have layer pointers as keys; each entry stores the left and right bound of the layer in the list format , e.g. [L3_min, L3_max]

x_vector - a vector that stores the left and right bounds of the cortical column along x dimension

y_vector - a vector that stores the left and right bounds of the cortical column along y dimension

storeSoma - specifies whether soma positions have to be stored in the output array (default is set to True).

cellBodiesOverlap - boolean value which defines whether cell somata can overlap; default is set to True;

cellDiameterArray - optional dictionary of cell model diameters required when cellBodiesOverlap is set to False;

This method returns the dictionary; each key is a unique cell population id and the corresponding value is a dictionary which refers to libNeuroML population object (key 'PopObj') and cell position array ('Positions') which by default is None.

opencortex.utils.**build_connectivity**(*net*, *pop_objects*, *path_to_cells*, *full_path_to_conn_summary*, *pre_segment_group_info=[]*, *return_cached_dicts=True*, *synaptic_scaling_params=None*, *synaptic_delay_params=None*, *distance_dependence_params=None*, *ignore_synapses=[]*)

This method calls the appropriate build and utils methods to build connectivity of the NeuroML2 cortical network. Input arguments are as follows:

net- NeuroML2 network object;

pop_objects - dictionary of population parameters in the format returned by the utils method add_populations_in_rectangular_layers() or add_populations_in_cylindrical_layers();

path_to_cells - dir path to the folder where target NeuroML2 .cell.nml files are found;

full_path_to_conn_sumary - full path to the file which stores the connectivity summary, e.g. file named netConnList in the current working dir, then this string must be "netConnList";

pre_segment_group_info - input argument of type 'list' which specifies presynaptic segment groups; made to supplement connectivity summary of type netConnList in the Thalamocortical project; default value is []; alternatively it might have one value of type'dict' or several values of type 'dict'; in the former case, dict should contain the fields 'PreSegGroup' and 'ProjType'; in the latter case each dictionary should contain the fields 'PrePop', 'PostPop', 'PreSegGroup' and 'ProjType', which uniquely specifies one presynaptic segment group per pair of cell populations per type of projection.

return_cached_dicts - boolean-type argument which specifies whether build_connectivity returns the cached dictionary of cummulative distributions of segment lengths for all of the target segment groups. If return_cached_dicts is set to True the last output argument that is returned by build_connectivity is a cached target dictionary; the cached target dictionary is specifically built by the method check_cached_dicts inside utils;

synaptic_scaling_params - optional input argument, default value is None. Alternatively, it takes the format of

[{'weight':2.0,'synComp':'all'}] or

[{'weight':2.5,'synComp':'GABAA','synEndsWith':[],'targetCellGroup':[]},
{'weight':0.5,'synComp':'Syn_Elect_DeepPyr_DeepPyr','synEndsWith':[],'targetCellGroup':['CG3D_L5']}
]. Tailored for the NeuroML2 Thalamocortical project.

synaptic_delay_params - optional input argument, default value is None. Alternatively ,it takes the format similar to the synaptic_scaling_params:

[{'delay':0.05,'synComp':'all'}] or

[{'delay':0.10,'synComp':'GABAA','synEndsWith':[],'targetCellGroup':[]},
{'delay':0.05,'synComp':'Syn_Elect_DeepPyr_DeepPyr','synEndsWith':[],'targetCellGroup':['CG3D_L5']}
]. Tailored for the NeuroML2 Thalamocortical project.

distance_dependent_params - optional input argument, default value is None. Alternatively, it take the format of

[{'PrePopID':'Pop1','PostPopID':'Pop2','DistDependConn':'- 17.45 + 18.36 / (math.exp((r-267.)/39.) +1)','Type':'Elect'}].

opencortex.utils.**build_inputs**(*nml_doc*, *net*, *population_params*, *input_params*, *cached_dicts=None*, *path_to_cells=None*, *path_to_synapses=None*)

a wrapper method that calls appropriate methods to build inputs to the NeuroML2 network. Input arguments:

nml_doc - a libNeuroML doc object;

net - a libNeuroML net object;

population_params - a dictionary that stores population parameters in the format returned by the method add_populations_in_layers;

input_params -a dictionary that specifies input parameters for any given cell model. The format can be checked by the method check_inputs. Dictionary values must

be of type 'list' and can thus define multiple input groups on any given cell type. Examples where lists contain only one input group but differ in other parameters:

Example 1: input_params={'TCR':[{'InputType':'GeneratePoissonTrains',
'InputName':'TransPoiInputs', 'TrainType':'transient', 'Synapse':'Syn_AMPA_L6NT_TCR', 'AverageRateList':[0.05], 'DurationList':[200.0], 'DelayList':[20.0], 'FractionToTarget':1.0, 'LocationSpecific':False, 'TargetRegions':[{'XVector':[2,12],'YVector':[3,5],'ZVector':[0,5]}], 'TargetDict':{'dendrite_group':1000 } }] }

Example 2: input_params={'TCR':[{'InputType':'PulseGenerators', 'InputName':'PulseGenerator0', 'Noise':False, 'AmplitudeList':[20.0,-20.0], 'DurationList':[100.0,50.0], 'DelayList':[50.0,200.0], 'FractionToTarget':1.0, 'LocationSpecific':False, 'TargetDict':{'dendrite_group':2 } }] }

Example 3: input_params={'CG3D_L23PyrRS':[{'InputType':'PulseGenerators', 'InputName':'PulseGenerator0', 'Noise':True, 'SmallestAmplitudeList':[5.0E-5], 'LargestAmplitudeList':[1.0E-4], 'DurationList':[20000.0], 'DelayList':[0.0], 'TimeUnits':'ms', 'AmplitudeUnits':'uA', 'FractionToTarget':1.0, 'LocationSpecific':False, 'UniqueTargetSegmentID':0, 'UniqueFractionAlong':0.5 }] } ;

cached_dicts - optional argument, default value is set to None; otherwise should be the input variable that stores the dictionary in the format returned by check_cached_dicts;

path_to_cells - dir where NeuroML2 cell files are found;

path_to_synapses - dir where NeuroML2 synapse files are found.

opencortex.utils.**build_probability_based_connectivity**(*net*, *pop_params*, *probability_matrix*, *synapse_matrix*, *weight_matrix*, *delay_matrix*, *tags_on_populations*, *std_weight_matrix=None*, *std_delay_matrix=None*)

'Method which build network projections based on the probability matrix which specifies the probabilities between given populations. Input arguments:

net- NeuroML2 network object;

pop_params - dictionary of population parameters in the format returned by the utils method add_populations_in_rectangular_layers() or add_populations_in_cylindrical_layers();

probability_matrix - n by n array (list of lists or numpy array) which specifies the connection probabilities between the presynaptic and postsynaptic populations; the first index is for the target population (postsynaptic) and the second index is for the source population (presynaptic); can be 1 by 1 matrix , then probability is applied to all pairs of populations; probability values must be of type 'float';

synapse_matrix - n by n array (list of lists) which specifies the synapse components per projections; each element has be of the type 'list' because generically physical projection can contain multiple synaptic components;

weight_matrix - n by n array (list of lists or numpy array) which specifies the weight values for projections; each matrix element should be of type 'float' or if synaptic components per given projection differ in weights, then a corresponding matrix element must be a list containing 'float' values;

delay_matrix - n by n array (list of lists or numpy array) which specifies the delay values for projections; each matrix element should be of type 'float' or if synaptic components per given projection differ in delays, then a corresponding matrix elment must be a list containing 'float' values;

tags_on_populations - a list of n strings which tags the network populations; cannot have length larger than the number of populations; index of the population tag in tags_on_populations must correspond to the position of the matrix element in a standard way, e.g.

tags_on_populations= ['pop1', 'pop2'], thus 'pop1' index is 0 and 'pop2' index is 1;

source population (presynaptic)

target population (postsynaptic) 'pop1' 'pop2'

'pop1' (0,0) value in matrix (0,1) value in matrix

'pop2' (1,0) value in matrix (1,1) value in matrix . This applies to all matrices.

std_weight_matrix - optional matrix in the format weight_synapse which specifies the corresponding standard deviations of synaptic weights; default is set to None;

std_delay_matrix - optional matrix in the format delay_synapse which specifies the corresponding standard deviations of synaptic delays; default is set to None.

opencortex.utils.**build_projection**(*net*, *proj_counter*, *proj_type*, *presynaptic_population*, *postsynaptic_population*, *synapse_list*, *targeting_mode*, *pre_seg_length_dict*, *post_seg_length_dict*, *num_of_conn_dict*, *distance_dependent_rule=None*, *pre_cell_positions=None*, *post_cell_positions=None*, *delays_dict=None*, *weights_dict=None*)

This method calls the appropriate methods that construct chemical or electrical projections. The input arguments are as follows:

net - the network object created using libNeuroML (neuroml.Network());

proj_counter - stores the number of projections at any given moment;

proj_type - 'Chem' or 'Elect' depending on whether the projection is chemical or electrical.

presynaptic_population - object corresponding to the presynaptic population in the network;

postsynaptic_population - object corresponding to the postsynaptic population in the network;

synapse_list - the list of synapse ids that correspond to the individual receptor components on the physical synapse, e.g. the first element is the id of the AMPA synapse and the second element is the id of the NMDA synapse; these synapse components will be mapped onto the same location of the target segment;

targeting_mode - specifies the type of projection: divergent or convergent;

pre_seg_length_dict - a dictionary whose keys are the ids of presynaptic segment groups and the values are dictionaries in the format returned by make_target_dict();

post_seg_length_dict - a dictionary whose keys are the ids of target segment groups and the values are dictionaries in the format returned by make_target_dict();

num_of_conn_dict - a dictionary whose keys are the ids of target segment groups with the corresponding values of type 'int' specifying the number of connections per tarrget segment group per each cell.

distance_dependent_rule - optional string which defines the distance dependent rule of connectivity - soma to soma distance must be represented by the string character 'r';

pre_cell_positions- optional array specifying the cell positions for the presynaptic population; the format is an array of [x coordinate, y coordinate, z coordinate];

post_cell_positions- optional array specifying the cell positions for the postsynaptic population; the format is an array of [x coordinate, y coordinate, z coordinate];

delays_dict - optional dictionary that specifies the delays (in ms) for individual synapse components, e.g. {'NMDA':5.0} or {'AMPA':3.0,'NMDA':5};

weights_dict - optional dictionary that specifies the weights for individual synapse components, e.g. {'NMDA':1} or {'NMDA':1,'AMPA':2}.

opencortex.utils.**check_cached_dicts**(*cell_component*, *cached_dicts*, *list_of_target_seg_groups*, *path_to_nml2=None*)

This method checks whether information is missing on target segment groups and updates the output dictionary with new information for the target cell component.

opencortex.utils.**check_delay_params**(*delay_params*)

opencortex.utils.**check_includes_in_cells**(*dir_to_cells*, *list_of_cell_ids*, *extra_channel_tags=None*)

opencortex.utils.**check_inputs**(*input_params*, *popDict*, *path_to_cells*, *path_to_synapses=None*)

opencortex.utils.**check_matrix_size_and_type**(*matrix_of_params*, *num_of_pop_tags*, *type_of_matrix*)

Method to check whether the size and the type of the connection parameter matrix, corresponds to the number of tags provided for the list of populations.

opencortex.utils.**check_pop_dict_and_layers**(*pop_dict*, *boundary_dict*)

opencortex.utils.**check_pre_segment_groups**(*pre_segment_group_info*)

opencortex.utils.**check_segment_group**(*segment_groups*, *target_segment_group*)

opencortex.utils.**check_synapse_location**(*synapse_id*, *pathToSynapses*)

opencortex.utils.**check_weight_params**(*weight_params*)

opencortex.utils.**get_segment_groups**(*cell_id*, *path_to_cells*)

opencortex.utils.**parse_delays**(*delays_params*, *post_pop*, *synapse_list*)

opencortex.utils.**parse_distance_dependence_params**(*distance_dependence_params*, *pre_pop*, *post_pop*, *proj_type*)

opencortex.utils.**parse_parameter_value**(*parameter_matrix*, *row_ind*, *col_ind*, *checks_passed=False*)

Method to parse one of the connectivity parameters; by default assumes that checks are carried out by the method check_matrix_size_and_type().

opencortex.utils.**parse_weights**(*weights_params*, *post_pop*, *synapse_list*)

opencortex.utils.**read_connectivity**(*pre_pop*, *post_pop*, *path_to_txt_file*, *ignore_synapses=[]*)

Method that reads the txt file in the format of netConnList found in: Thalamocortical/neuroConstruct/pythonScripts/netbuild.

opencortex.utils.**replace_cell_types**(*net_file_name*, *path_to_net*, *new_net_id*, *cell_types_to_be_replaced*, *cell_types_replaced_by*, *dir_to_new_components*, *dir_to_old_components*, *reduced_to_single_compartment=True*, *validate_nml2=True*, *return_synapses=False*, *connection_segment_groups=None*, *input_segment_groups=None*, *synapse_file_tags=None*)

This method substitutes the target cell types to a given NeuroML2 cortical network.

Python Module Index

o

Index