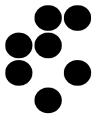
Jozef Stefan Institute
Department of Communication Systems
Jamova 39, SI-1000 Ljubljana, Slovenia



IJS Delovno poročilo

DP- 12298

Localization Toolbox: User manual

Version 1.0

Tomaž Javornik, Andrej Hrovat, Igor Ozimek, Roman Novak, Mihael Mohorčič

Ljubljana, Februar, 2017

Welcome to Localization Toolbox's documentation!

Exp_Localization_00:

is an example how to use a toolbox to build an experiment

Experiment objective:

test a localisation algorithms in a real outdoor environment

Building experiment consists of following steps:

- *setup of radio environment*
- *reading or generation of the measurements*
- *running experiments*
    - *LS localization*
    - *finger printing localization*

Version 0.0: Tomaz Javornik (January 2017)

RE.Radio_Env

contains the definition of the radio environment class.

class **RE.Radio_Env.RadioEnvironment**(*Id*) [source]

Class stores and process radio environment:

append(*typ, Id, Element*) [source]

Appends a new radio environment element.

Parameters:
- *typ (str) – type of radio environment ["Network", "Raster Map", "Measurements"]*
- *Id (str) – Id of imported element*
- *Element (radio_network,) – element to be imported*

Returns:

IJS_Outdoor.append("Network", "Anchors", Anchors)

delete(*typ, Id*) [source]

Deletes the radio element from the radio environment.

Parameters:
- *typ (str) – type of radio environment*
- *Id (str) – Id of radio environment*

Returns:

experiment_est_loc_FP(*pl_exp, ref_node_index, hist_plot*) [source]

Describes experiment in radio environment:

estimates the agent locations based using finger print method

Parameters:
- *pl_exp (float) – path loss exponent to convert RSSI to distance*
- *ref_node_index (int) – index of the reference node*
- *hist_plot (bool) – flag for plotting histogram of errors*

Returns:

experiment_est_loc_LS(*pl_exp, ref_node_index, hist_plot*) [source]

Describe experiment in radio environment:

Estimates the agent locations using least square method.

Parameters:
- *pl_exp (float) – path loss exponent to convert RSSI to distance*
- *ref_node_index (int) – index of the reference node*
- *hist_plot (bool) – flag for plotting histogram of errors*

Returns:

get(*typ*, *Id*) [source]

Returns a radio environment element.

Parameters:
- *typ (str) – type of radio environment*
- *Id (str) – Id of radio environment*

Returns: *radio environment element*

Return type: *network, raster_maps, measurements*

IJS_Outdoor.get("Network", "Anchors")

get_Ids(*typ*) [source]

Returns a list of radio environment Ids of type typ.

Parameters: *typ (str) – type of radio environment ["Network", "Raster Map", "Measurements"]*
Returns: *a list of radio environment of type typ*
Return type: *[str, str, str, ..]*

get_Region() [source]

Returns: *the region of the radio environment [west, south, delta_west, delta_north, cols, rows]*

replace(*typ*, *Id*, *element*) [source]

Replaces the element in radio environment.

Parameters:
- *typ (str) – element type*
- *Id (str) – element Id*
- *element – new element*

Returns:

set_Region(*Region*) [source]

Sets the region of the radio environment.

Parameters: *float, float, float, int, int Region (float,) – [west, south, delta_west, delta_north, cols, rows]*
Returns:

IJS_Outdoor.set_Region([0.0, 0.0, 1.0, 1.0, 10, 10])

RE.Radio_Net

package contains node and network class.

RE.Radio_Net.RSSI_to_dist(*RSSI_dBm*, *fc_MHz*, *tx_pow_dBm*, *pl_coef*) [source]

Converts measured RSSI level to the distance using FSPL channel model.

Parameters:
- *RSSI_dBm (float) – RSSI value in [dBm]*
- *fc_MHz (float) – carrier frequency in [MHZ]*
- *tx_pow_dBm (float) – transmit power in [dBm]*
- *pl_coef (float) – path loss coefficient*

Returns: *distance in [m]*

Return type: *float*

class **RE.Radio_Net.RadioNetwork**(**args*) [source]

Defines radio network class: a set or radio nodes

add_rnd_Loc(*Region*, *N_nodes*) [source]

Adds nodes random locations.

Parameters: *Region – region of the map*

- *N_nodes – number of nodes*

Returns:

append_RadioNode(*radionode*) [source]

Adds the RadioNode Node to the network.

Parameters: *Node – RadioNode*
Returns:

est_loc_FP(*radio_env, Measurements_Id, Anchors_Id, pl_exp, ref_anchor_index*) [source]

Estimate locations of nodes in radio network applying finger printing method.

Parameters:
- *radio_env – radio environment*
- *Measurements_Id – id of measurements*
- *Anchors_Id – id of anchor network*
- *pl_exp – pathloss exponent*
- *ref_anchor_index – reference anchor index*

Returns:

est_loc_LS(*radio_env, Measurements_Id, Anchors_Id, pl_exp, ref_anchor_index*) [source]

Estimates locations of nodes in radio network applying least square method.

Parameters:
- *radio_env – radio environment*
- *Measurements_Id – id of measurements*
- *Anchors_Id – id of anchor network*
- *pl_exp – pathloss exponent*
- *ref_anchor_index – reference anchor index*

Returns:

get_AreaLimits(*fract*) [source]

Finds the rectengular area, where nodes are located.

Parameters: *fract – fraction of area to extend display*
Returns: *[west, south, east, north]*

get_Id() [source]

Returns: *network Id*

get_Len() [source]

Returns: *number of nodes in radio network*

get_Locations() [source]

Returns node locations of nodes in network.

Parameters: *nodeId – nodes Id*
Returns: *[west-east, south-north, alatitude above ground level]*

get_RadioNode(*NodeId*) [source]

Returns RadioNode with NodeId.

Parameters: *NodeId – Id of radion node*
Returns: *RadioNode*

get_RadioNode_FcMHz(*nodeId*) [source]

Returns RadioNode carrier frequency.

Parameters: *nodeId – node Id*
Returns: *carrier frequncy in MHz*

get_RadioNode_Ids() [source]

Returns list of RadioNode Ids.

Returns: *list of node Ids*

get_RadioNode_Index(*NodeId*) [source]

Returns radio node index

Parameters: *NodeId – Id of radio node*
Returns: *RadioNode*

get_RadioNode_Loc(*nodeId*) [source]

Returns node location of node.

Parameters: *nodeId – nodes Id*
Returns: *[west-east, south-north, alatitude above ground level]*

get_RadioNode_PowdBm(*nodeId*) [source]

Returns RadioNode power.

Parameters: *nodeId – node Id*
Returns: *Tx/Rx power of node in dBm*

get_RadioNodes() [source]

Returns: *list of network Ids*

get_Region_Map(*margin*) [source]

Returns radio map region.

Parameters: *margin – margin beyond the area limits*
Returns: *region map [west, south, delta_west, delta_south, cols, rows]*

loc_error(*ref_network, plothist, D3*) [source]

Esitmates location error between refence network and radio network.

Parameters:
- *ref_network – reference network*
- *plothist – flag for printng error histogram*
- *D3 – flag to estimate error in three dimensions*

Returns: *[mean, median, error standard deivation]*

plot(*Marker, FigNum, ShowPlot, delta*) [source]

Plots the radio network on the map background.

Parameters:
- *Marker – marker*
- *FigNum – figure number*
- *ShowPlot – show plot flag*
- *delta – size of boarder*

Returns:

print_Net() [source]

Prints the radio nodes on console.

Returns:

read_fromCsvFile(*csvFilename*) [source]

Reads newtwork from csv file.

Parameters: *csvFilename –*
Returns:

set_Id(*Id*) [source]

Sets network Id

Parameters: *Id (str) – network id*
Returns:

set_RadioNode_Alt(*nodeId, alt*) [source]

Sets radio node altitide above sea level.

Parameters:
- *nodeId – node Id*
- *alt – altitude in [m]*

Returns:

set_RadioNode_BwMHz(*nodeId, Bw*) [source]

Sets radio node bandwidth.

Parameters:
- *nodeId – node Id*
- *Bw – bandwidth in MHz*

Returns:

set_RadioNode_FcMHz(*nodeId, Fc*) [source]

Sets radio node carrier frequency.

Parameters:
- *nodeId – node Id*
- *Fc – node carrie frequency in MHz*

Returns:

set_RadioNode_Loc(*nodeId, Loc*) [source]

Sets locations of radio nodes.

Parameters:
- *nodeId – node Id*
- *Loc – node location*

Returns:

set_RadioNode_PowdBm(*nodeId, Pow*) [source]

Sets radio node Tx/Rx power.

Parameters:
- *nodeId – node Id*
- *Pow – bandwidth in dBm*

Returns:

class RE.Radio_Net.**RadioNode**(**args*) [source]

Defines RadioNode class.

est_loc_FP(*radio_env, Measurements_Id, Anchors_Id, Mobiles_Id, pl_coef, ref_anchor_index*) [source]

This is method returns a location of node using fingerprint method.

Parameters:
- *radio_env – radio environment*
- *Measurements_Id – Id of measurements*
- *Anchors_Id – Id of Anchor network*
- *Mobiles_Id – Id of Mobile network*
- *pl_coef – Id of path loss coefficient*
- *ref_anchor_index – index of reference Anchor node*

Returns: *location x, y, z coordinates of node*

est_loc_LS(*radio_env, Measurements_Id, Anchors_Id, Mobiles_Id, pl_coef, ref_anchor_index*) [source]

This method estimates the location of node using least square method.

Parameters:
- *radio_env – radio environment*
- *Measurements_Id – Id of measurements*
- *Anchors_Id – Id of Anchor network*
- *Mobiles_Id – Id of Mobile network*
- *pl_coef – Id of path loss coefficient*
- *ref_anchor_index – index of reference Anchor node*

Returns: *location x, y, z coordinate of the node*

get_Alt() [source]

Returns: *altitude above see level in [m]*

get_AntAvgl() [source]

Returns: *altitude above ground level in [m]*

get_AntAzimuth() [source]

Returns: *antenna azimuth in degrees*

get_AntTilt() [source]

Returns: *antenna tilt in degrees*

get_AntType() [source]

Returns: antenna type

get_BwMHz() [source]

Returns: node bandwidth in MHz

get_FcMHz() [source]

Returns: node carrier frequency in MHz

get_Id() [source]

Returns: Node Id

get_LatLong() [source]

Returns: node location in WGS84 format lat, long

get_Loc() [source]

Returns: node location

get_Name() [source]

Returns: node name

get_PowdBm() [source]

Returns: transmit power in dBm if Tx, Rx power in dBm if Rx

get_WGS84() [source]

Returns: node location in WGS84 format [lat, long, altitude above see level]

get_Z() [source]

Returns: altitude above ground level in [m]

print_node() [source]

Prints node configuration.

Returns:

set_Alt(*alt*) [source]

Sets alititude above the see level in [m].

Parameters: alt – altitude in [m]
Returns:

set_AntAvgl(*alt*) [source]

Sets node altitude above the ground level.

Parameters: alt –
Returns:

set_AntAzimuth(*azimuth*) [source]

Sets antenna azimuth in degrees.

Parameters: azimuth – antenna azimuth in degreee
Returns:

set_AntTilt(*tilt*) [source]

Sets antenna tilt.

Parameters: tilt – antenna tilt in degrees
Returns:

set_AntType(*ant*) [source]

Sets antenna type.

Parameters: ant – antenna type
Returns:

set_BwMHz(*bw*) [source]

Sets node frequency bandwidth in MHz.

Parameters: *bw – frequency bandwidth in MHz*
Returns:

set_FcMHz(*fc*) [source]

Sets carrier frequency.

Parameters: *fc – carrier frequency in MHz*
Returns:

set_LatLong(*xlat*, *xlong*) [source]

Sets node location in WGS 84 format.

Parameters:
- *xlat – latitude in WGS84 dec format*
- *xlong – longitude in WGS84 dec format*

Returns:

set_Loc(*loc*) [source]

Sets node location.

Parameters: *loc – location in [x, y, z] format*
Returns:

set_Name(*Name*) [source]

Sets node name.

Parameters: *Name – node Name*
Returns:

set_PowdBm(*p*) [source]

Sets Rx or Tx power in dBm.

Parameters: *p – power in dBm*
Returns:

set_XY(*x*, *y*) [source]

Sets node location.

Parameters:
- *x – node x (east - west)*
- *y – node y (south - north)*

Returns:

set_Z(*alt*) [source]

Sets node altitude above the ground level.

Parameters: *alt –*
Returns:

to_WGS84() [source]

Calculates lat, long from Guass Kreuger coordinates and set it in self.WGS84.

Returns:

RE.Radio_Net.**to_num**(*x*) [source]

Converts string to number.

Parameters: *x (str) – string to convert*
Returns: *number*
Return type: *number*

RE.Radio_Net.**to_str**(*x*) [source]

Converts number to sting.

Parameters: *x (float) – number*
Returns: *string of x*
Return type: *str*

! !!!!!! ! !!

contains:

- *RasterMap class*
- *RasterMaps class, a set of raster maps*

class **RE.Ra□ter_Ma□.RasterMap**(*Id, west, south, delta_west, delta_south, cols, rows*) [source]

Defines raster map object

ColsRows_to_xy(*cols, rows*) [source]

Converts map column, map row location to east-west, south-east location.

Parameters:
- *cols – map column*
- *rows – map row*

Returns: *east-west, south-east location*

RSSI_FSPL(*node*) [source]

Calculates the FSPL map for the transmitter at node.

Parameters: *node – Tx node*

Returns: *FSPL map*

add_Random(*Distribution, Params*) [source]

Adds random number to the map pixel values.

Parameters:
- *Distribution – distribution ["Normal", "Rayleigh", "LogNormal"]*
- *Params – distrubution parameters*

Returns:

copy(*MapId*) [source]

Copies the map.

Parameters: *MapId – Id of new map*

Returns: *new map*

dist(*x_point, y_point, val_point*) [source]

Returns map distance from point (x,y,val).

Parameters:
- *x_point – east-west coordinate*
- *y_point – south-east coordinate*
- *val_point – value*

Returns:

generate_xy_mesh() [source]

Generates the mesh grid of west-east and south north values.

Returns: *X -> west-east, Y -> south - north array of raster points (x, y)*

get_Id() [source]

Returns: *map Id*

get_Region() [source]

Returns: *region [west, south, d_west, d_south, Cols, Rows]*

get_Region_WSEN() [source]

Returns: *region in west, south, east, north format*

get_Values() [source]

Returns: *the values of raster map*

get_aValue(*x, y*) [source]

Finds the value of map at [x, y] location.

Parameters:
- *x – east-west*
- *y – south-north*

Returns: *raster map value*

plot(*FigNum, ShowPlot, ColorBar, Region*) [source]

Plots raster map.

Parameters:
- *FigNum – figure number*
- *ShowPlot – flag to show plot on the screen*
- *ColorBar – flag to show color bar*
- *Region – map region*

Returns:

plot_Network(*FigNum, ShowPlot, Marker, MarkerSize, Network, ColorBar, Legend_label, Region*) [source]

Plots the network.

Parameters:
- *FigNum – figure number*
- *ShowPlot – True/False if we want plot is shown*
- *Marker – marker type ["s", "o", "x", ...]*
- *MarkerSize – size of marker in a plot*
- *Network – network to be plotted*
- *ColorBar – True/False plot color bar*
- *Legend_label – Legend labels at nodes*
- *Region – map region*

Returns:

print_Region() [source]

Prints the region of the map.

Returns:

read_from_xyz_RaPlaT(*csv_file, delimiter, not_a_number*) [source]

Imports Grass/RaPlaT file na generate raster map.

Parameters:
- *ascii_file – ascii file from Grass/RaPlaT toolbox*
- *delimiter – delimiter between data*

Param: *not_a_number: sign for not a number value*

Returns: *raster map*

set_Id(*MapId*) [source]

Sets map Id.

Parameters: *MapId – map Id*

Returns:

set_Values(*in_values*) [source]

Sets values to raster map.

Parameters: *in_values – np array of values*

Returns:

set_aValue(*x, y, value*) [source]

Sets a raster map value at coordinate x, y.

Parameters:
- *x – west-east coordinate*
- *y – south-north coordinate*
- *value – value*

Returns:

xy_to_ColsRows(*x, y*) [source]

Converts east-west, south-east location to map column and map row.

Parameters:
- *x – east-west coordinate*
- *y – south-east coordinate*

Returns: *column, row*

class RE.Ra□ter_Ma□.**RasterMaps**(**args*) [source]

Defines a set of raster maps.

add_Random(*Distribution, Params*) [source]

Adds a random noise to the map.

Parameters:
- *Distribution – distribution*
- *Params – distribution parameters*

Returns:

append_Map(*Map*) [source]

Adds a map to the maps.

Parameters: *Map – raster map*

Returns:

calc_RSSI_FSPL(*Region, Txs*) [source]

Calculates received signal level using FSPL channel model.

Parameters:
- *Region – set region*
- *Txs – set of transmitters*

Returns:

get_Id() [source]

Returns: *raster maps Id*

get_Length() [source]

Returns a number of maps in raster_maps.

Returns: *umber of maps*

get_Map(*MapId*) [source]

Gets a map with MapId from the maps.

Parameters: *MapId – map id*

Returns:

get_Net() [source]

Returns: *network associated with the map*

get_Region() [source]

Returns a Region map of maps.

Returns:

get_Values(*x, y*) [source]

Returns values of all maps in a set at location x, y.

Parameters:
- *x – east-west location*
- *y – north-south location*

Returns:

set_Id(*Id*) [source]

Sets maps Id.

Parameters: *Id – maps Id*

Returns:

set_Net(*network*) [source]

Sets network associated with the maps.

Parameters: *network – network Id*

Returns:

set_Region(*Region*) [source]

Sets a Region map of maps.

Parameters: *Region – region map*

Returns:

set_Values(*x*, *y*, *value*) [source]

Sets a value to all maps in a set.

Parameters:
- *x – east-west location*
- *y – north-south location*
- *value – value*

Returns:

! !!!!!!! !!!

contains the definition of measurement classes: - measurement - trace - a set of measurements

class **RE.Mea□□re□ent□.Measurement**(*Id*) [source]

Measurement class stores and process measurements

get_Id() [source]

Returns: Id of measurement

get_Rx_Network_Id() [source]

Return id of rx network:

get_Rx_Node_Id() [source]

Returns: Rx node Id

get_TimeStamp() [source]

Returns: returns the time stamp of measurement

get_Tx_Network_Id() [source]

Returns: id of Tx network

get_Tx_Node_Id() [source]

Returns: Tx node Id

get_Type() [source]

Returns: type of measurement

get_Unit() [source]

Returns: unit of measurement

get_Value() [source]

Returns: value of measurement

print_Measure() [source]

Prints the measurement on the console.

Returns:

set_Id(*Id*) [source]

Sets Id of measurement.

Parameters: Id – measurement Id

Returns:

set_Rx_Network_Id(*x*) [source]

Sets id of Rx network.

Parameters: x – Rx network id

Returns:

set_Rx_Node_Id(*x*) [source]

Sets Rx node Id.

Parameters: x – Rx node Id

Returns:

set_TimeStamp(*x*) [source]

Setd time stamp of measurement.

Parameters: *x – time stamp of measurement*

Returns:

set_Tx_Network_Id(*x*) [source]

Sets od of Tx network.

Parameters: *x – Tx network id*

Returns:

set_Tx_Node_Id(*x*) [source]

Sets Tx node Id.

Parameters: *x – Rx node Id*

Returns:

set_Type(*x*) [source]

Sets type of measurement.

Parameters: *x – type of measurement*

Returns:

set_Unit(*x*) [source]

Sets unit of measurement.

Parameters: *x –*

Returns:

set_Value(*x*) [source]

Sets value of measurement.

Parameters: *x – value of measurement*

Returns:

class RE.Mea□□re□ent□.**Trace**(*Id*) [source]

Class defines a set of measurement.

add_error(**args*) [source]

add error to the measurement :param args: type of error :returns:

append(*x*) [source]

Appends measurement.

Parameters: *x – measurement in form of dictionary:*

Returns:

get(*tx_net_Id, tx_node_Id, rx_net_Id, rx_node_Id*) [source]

Gets all measurements which corresponds to following parameters:

Parameters:
- *tx_net_Id – tx network id*
- *tx_node_Id – tx node id*
- *rx_net_Id – rx network id*
- *rx_node_Id – rx node id*

Returns: *list of measurements*

get_Id() [source]

Returns: *trace id*

get_measure_index(*Id*) [source]

Parameters: *id – measurement Id*

Returns: *measurement index*

len() [source]

Returns: number of measurements

print_Trace() [source]

Prints trace on the console.

Returns:

quantize_vals(quant) [source]

Quantizes the measurement.

Parameters: quant – quantizaion of measurement results
Returns:

remove_measure(measure_Id) [source]

Removes measurement from trace.

Parameters: measure_Id – measurement id
Returns:

replace_measure(measure_Id, x) [source]

Replaces measurement in trace.

Parameters:
- measure_Id – measurement id
- x – new measurement

Returns:

set_Id(Id) [source]

Sets trace id.

Parameters: Id – trace id
Returns:

set_measure_value(measure_Id, value) [source]

Replaces the value of measurements.

Parameters:
- measure_id – measurement id
- value – new value

Returns:

set_values_from_maps(Maps, Locs, Rx_Ids, Tx_Ids, x) [source]

Sets measurement value from the raster maps.

Parameters:
- Maps – list of raster maps
- Locs – list of locations
- Rx_Ids – list of Rx Ids
- Tx_Ids – listo of Tx Ids
- x – measurement

Returns:

! !! ! !! !!

package contains GIS functions.

□i□□.□□□.GKtoWGS84(x, y) [source]

Converts Gauss Kreuger EW/NS coordinate to latitude and longitude in WGS84 dec format

Parameters:
- x (float) – east-west value in Gauss Kreuger
- y (float) – south-north value in Gauss Kreuger

Returns: [latitude, longitude]

Return type: [str, str]

lat, lng = GKtoWGS84(x, y)

[illegible].[illegible].WGS84toGK(*in1, in2*) [source]

Converts latitude, longitude in WGS84 format to Gauss Kreuger coordinates

Parameters:
- *in1 (str) – latitude in dec format*
- *in2 (str) – longitude in dec format*

Returns: *Gauss Kreuger east-west, south-north coordinates*

Return type: *[float, float]*

x,y = WGS84toGK(14.50, 46.50)

[illegible].[illegible].dec2dms(*indeg*) [source]

Returns value written in decimal format in degrees, minutes, seconds format

Parameters: *indeg (float) – value in decimal format*
Returns: *value in degrees, minutes, seconds format*
Return type: *str*

[illegible].[illegible].dms2dec(*dms_str*) [source]

Returns decimal representation of DMS.

Parameters: *dms_str (str) – string in degrees minutes second format*
Returns: *value of dms_str in decimal format*
Return type: *float*

class **[illegible].[illegible].google_map**(*title, html_file, map_center, map_zoom, map_type*) [source]

Google maps object.

Creates a google_map object:

Parameters:
- *title (str) – title of page*
- *html_file (str) – filename of html_file*
- *float] map_center ([float,) – center of map*
- *map_zoom (float) – zoom of map*
- *map_type (str) – map type: "roadmap", "satellite", "hybrid,", "terrain"*

Returns: *google_map object*

html_map = google_map("LOG.a.TEC", plot_html_file, [46.0422, 14.4885], 20, "road")

add_net_from_csv(*csvfile, network_name, marker*) [source]

Adds network defined by csv file to the google_map object

Parameters:
- *csvfile (str) – csv file name with the network*
- *network_name (str) – name of the netwok*
- *marker (str) – marker of nodes*

Returns:

google_map.add_net_from_csv(csv_file, "Anchors", "sg")

add_network(*network, marker*) [source]

Adds a network network to the google_map object

Parameters:
- *network (str) – name on network*
- *network_name (str) – name of the network*
- *marker – marker type*

Returns:

add_node(*node_Id, position, info_str, marker*) [source]

Adds new location (node) on the google map

Parameters:
- *node_Id (str) – Id of node*
- *float] position ([float,) – [latitude, longitude] in dms format*
- *str,] info_str ([str,) – display at clicking on the node*
- *marker (str) – node marker: [x,s,o,f,a][r,g,b,k], "xg"*

Returns:

html_map.add_node(4, [46.0423773, 14.4882363], ["Node 4", "text 4", "text 4"], "ak")

add_overlay(*overlay_Id, figure, bounds, opacity*) [source]

Adds the transparent overlay on the google map object

Parameters:
- *overlay_Id (str) – overlay ID*
- *figure (str) – figure file of the overlay*
- *float, float] bounds ([float,) – bounds of figure: [north, south, east, west]*
- *opacity (float) – opacity [0 - 1.0] [transparent, non-transparent]*

Returns:

google_map.add_overlay(1, "test.png", region, 0.5)

get_file_name() [source]

Returns html file name of the google_map object

Returns: *name of html file*
Return type: *str*

get_number_of_nodes() [source]

Returns number of nodes in google_map object

Returns: *number of nodes*
Return type: *int*

[illegible].is_LatLong(*s*) [source]

Returns float value of s: positive for N, E; negative for W, S

Parameters: *s (str) – latitude/longitude in dec format*
Returns: *float(s): positive for N, E; negative for W, S*
Return type: *float*

[illegible]

package contains routines for JSON file.

[illegible].json_show_Google_Maps(*infile, map_title, html_file, map_zoom, map_type, marker*) [source]

Reads radio network form JSON file and generates html file with nodes location.

Parameters:
- *infile (str) – json file with network description*
- *map_title (str) – title of the map*
- *html_file (str) – html file name*
- *map_zoom (str) – google maps zoom parameter*
- *map_type (str) – type of the map: roadmap, satellite, hybrid, terrain*
- *marker (str) – type of marker: o, s, x, a, f; colour of marker: r, g, b, k*

Returns:

[illegible].json_to_RaPlaTcsv(*infile, outfile, radius, chan_param*) [source]

Reads json file and coverts it to the csv files for RaPlaT.

Parameters:
- *infile (str) – input json file*
- *outfile (str) – csv file names*
- *radius (str) – calculation radius*
- *chan_param (str) – channel parameters RaPlaT format*

Returns: *antenna Id*

Return type: *str*

[illegible]

initializes a set of project directories:

- *projDir directory*
- *dataDir: data directry*
- *figDir: figure directory*
- *progDir: program directory*

and data units, data types, google map types:

- *Data_Units = ["m", "km", "s", "ms", "dBm", "dBu"]*
- *Data_Types = ["RSSI", "ToA", "TDoA", "Dist"]*
- *Google_Map_Types = ["roadmap", "satellite", "hybrid,", "terrain"]*

i.ini.roet.get_proj_dir(*N*) *[source]*

Gets N-th ancestor director of the project dir.

Parameters: *N (int) – N-th ancestor directory: N = 0 project dir*
Returns: *path to current working directory*
Return type: *str*

i.ini.roet.iniDir(*projDir*) *[source]*

iniDir is used to specify directory structure of the project.

Parameters: *projDir (str) – path to project directory*
Returns:

- *Index*
- *Module Index*
- *Search Page*