latest change: April 3, 2019

Documentation for RaLaNS

This document gives an introduction to the main elements of the propagation loss model. *RaLaNS* [1] contains of two main parts, the ray launching, which can be used seperately, and the *ns-3* integration.

Table of Contents

1 Installing RaLaNS

1.1 Prerequisites

First install required packages for installation of *RaLaNS*. The given commands are suitable for installation via the packagemanagement system apt-get given standard *Ubuntu* distributions. For other *Unix* systems use equivalent commands for package installation.

```
sudo apt-get update
sudo apt-get install g++ python python-numpy python-matplotlib python-pp
   python-setuptools python-lxml subversion python-opengl python-pygame python
   -pip make
```

Installation of the utm package (for transformation of coordinates) is not possible through apt-get:

```
pip install utm
```

1.2 Installation

```
cd <pathToRaLaNS>/ralans/src/launcher
make
cd <pathToRaLaNS>/ralans
```

Problems

If there is the following Error:

```
#error This file requires compiler and library support for the ISO C++ 2011
   standard. This support is currently experimental, and must be enabled with
   the \texttt-std=c++11 or -std=gnu++1 compiler options.
```

Uncomment the compiler options behind `CFLAGS` in:
`<pathToRaLaNS>/ralans/src/launcher/Makefile`

1.3 Test the environment

Now you should be able to successfully use *RaLaNS*.

```
python run.py inputfiles/small_street_flat.osm name='example'
```

As soon as the process ended successfully you can see the results.

```
python viewer2d.py outputfiles/small_street_flat_example.zip
```

More usage examples can be seen in 7.

1.4 Problem handling

If there is an error with `multiarray` not found, you might have an old version of `pp`, make sure, you have the following version - On Ubuntu 14.04 or higher, this did not happen:

- pp (Parallel Python) 1.6.4

Try:

```
pip install pp
```

Or manual installation:

```
http://www.parallelpython.com/downloads/pp/pp-1.6.4.tar.gz
tar -xzf pp-1.6.4.tar.gz
cd pp-1.6.4
sudo python setup.py install
```

If other problems arise, check if you are using the following versions or higher:

- Python 2.7.4 (2.7.6 has been tested successfully, too)
- NumPy 1.7.1 (1.6.1 has been tested successfully, too)
- matplotlib 1.3.1 (1.4.3 has been tested successfully, too)
- Ubuntu 14.10 (13.04, 12.04 and LinuXMint 17.1 has been tested successfully, too)

2 Structure

2.1 Files

This main folder contains a software, that prepares a file with signal strengths for usage in *ns-3*.

2.1.1 run.py

This is the main software, specify a mapfile as first param OR `-g` to generate a sample configuration file. Set `-m` as second param to generate a zipfile with prepared mapdata for instant usage. All values in the sample configuration file are set by default. If you don't need other values, you don't have to specify a configuration file as command parameter. The weighting of the configuration values is the following: At first all default values will be restored. After that the script overwrites the default values with the values from the configuration file. The last step overwrites the current values with the values from the command line.

optional parameters

- `configurationfile` : File extension has to be '`cfg`', replaces default configuration with your configuration.
- `transmitterfile` : File extension has to be '`tr`', you can specify several transmitter positions.
- `receiverfile` : File extension has to be '`rec`', you can specify several receiver positions.
- `<key>=<value>` : Accepts every key in the configfile as commandline argument and replaces current configuration with the value given, make sure that you don't use spaces in one argument. A list with names and descriptions of these parameters is given in `/src/config.py`.

Usage

```
python run.py -g | mapfile [configfile] [transmitterfile] [receiverfile] [key=
    value]* | mapfile -m
```

2.1.2 viewer1d.py

Visualizes a signal strength trend between two points, specify the zipfile of your calculation and a point. Only usable for scenarios with receivertypes `line`, `area` or `cubic`.

optional parameters

- `t <transmitter>` : Loads coveragemap of specified transmitter [X,Y,Z].
- `<point>to<point>` : Displays the signal strength along a line between these specific points.
- `s <name>` : Saves plot as `<name>.png` and `<name>.pdf`, has to be the last param.

If you want to display several signal trends in one plot, specify another (or the same) result zipfile and a point.

Usage

```
python <input.zip> [t <transmitter>] <receiver>[to<reciever>] [<input.zip> [t
   <transmitter>] <receiver>[to<reciever>] ...] [s <name>]
```

OR if your receiver-type is a line:

```
python <input.zip> l [t <transmitter>] [s <name>]
```

2.1.3 viewer2d.py

Visualizes a signal strength map, specify a result zipfile as first parameter. Only for scenarios with receivertype `area` or `cubic`.

optional parameters

- `<transmitter>` : Loads coveragemap of specified transmitter [X,Y,Z].
- `<layer>` : Loads coveragemap of specified layer (only available if receiver simulation type is 'cubic').
- `s <name>` : Saves plot as `<name>.png` and `<name>.pdf`, has to be the last param.

If you want to visualize the difference between two coveragemaps, specify a second result zip file.

Usage

```
python viewer2d.py <input.zip> [transmitter] [layer] [<input2.zip [transmitter
   ] [layer]] [s <name>]
```

2.1.4 viewer3d.py

Visualizes the map and rays, specify a result zipfile as first argument.

optional parameters

- `<transmitter>` : Loads rays of specified transmitter [X,Y,Z].

Usage

```
python viewer3d.py <input.zip> [transmitter]
```

2.1.5 viewer[Circle]Scattered.py

Visualizes the signal strength points on the map, specify a result zipfile as first parameter. Only for scenarios with receivertype `list / street`.

optional parameters

- `<transmitter>` : Loads rays of specified transmitter [X,Y,Z].
- `-b` : Draws buildings.

Usage

```
python viewerCircleScattered.py <input.zip> [transmitter] [-b]
```

2.1.6 calculationServer.py

Run this program to receive a calculation task. **DO NOT** use this and run.py on the same computer.

2.2 Folders

▷ `src` : Contains all the modules that are used by the software mentioned above.

▷ `inputfiles` : Put maps in here.

▷ `configfiles` : Configure the ray-launching with these files.

▷ `ouputfiles` : The outputfile that can be used in ns-3 will be put here.

▷ `tmp` : Auto-generated folder. All generated data for a scenario will be found here.

3 Additional information for the main program

3.1 Logging

`run.py` supports five different logging level, which affects, what is stored in the result zipfile. Higher level contains all files of lower levels, specify the level in your configuration by using the key `debugLevel`.

1 : zipfile contains configuration-files and the result-file for ns3

2 : adds needed files for 2d-viewer

3 : adds needed files for 3d-viewer

4 : stores the whole tmp-folder

5 : enables debug messages in logfile (might be interesting for developers)

debugRays: `True` - adds files to draw rays in 3d-Viewer

3.2 Parallel Calculation

If you calculate the signal for several transmitter positions, it can be performed parallel:

- specify the number of cpu-cores under the configuration-key '`numberWorkers`'

You can also calculate the result on several computers. They will be used if you want to calculate rays for more transmitter positions than you have cpu-cores.

- specify the ip-addresses of those computers under the configuration-key '`calculationServers`'
- each of those computers has to run `calculationServer.py`

3.3 Map-Files

Currently supported are the following formats:

- City-GML. Please rename your file to `<map>.gml`
- Openstreetmap. Please rename your file to `<map>.osm`
- Files containing polygons. The last and first point should be the same. Please use the file extension '`raw`'.

3.4 Simulation Types

Simulation types are used to describe transmitter and receiver positions.

0 : Point, x y z

1 : Line, xstart ystart zstart xend yend zend steps

2 : Area, xmin ymin xmax ymax z xstep ystep

3 : Cubic, xmin ymin zmin xmax ymax zmax xstep ystep zstep

4 : List, size x0 y0 z0 ... xn yn zn

3.4.1 Transmitter positions

- if you want to calculate signals for one transmitter (default), set your transmitter position under the configuration-key '`transmitters`', default is `transmitters=[[0,0,1]]`
- if you want to calculate signals for several transmitter, which are ordered in a line, specify a transmitter-file as command argument
- if you want to calculate signals for all transmitter positions on your map, set the configuration-key '`transmitters`' to '`area`'. It generates transmitter positions all over the map, the distance between each transmitter is specified in configuration-key '`stepSize`', the configuration-key '`coverageLevel`' describes the height of each transmitter.
- for 3d coverage of your transmitters, set the configuration-key '`transmitters`' to '`cubic`'. It generates transmitter positions all over the map and between '`coverageLevel`' and '`coverageMaxLevel`' and with distance between each position specified in '`stepSize`'.
- if you want to calculate signal for transmitters on specific points, you can either specify them under the configuration-key '`transmitters`' (`transmitters=[[0,0,1],[1,1,1],...]`) or in a file as command argument

3.4.2 Receiver positions

- if you want to receive the signal strength in one point, set your receiver position under the configuration-key '`receivers`' (for example: `receivers=[[0,0,1]]`)
- if you want to place receivers along a line, specify a line in a file and set it as command argument
- if you want to cover a whole area, set the configuration-key '`receivers`' to '`auto`' (default)
- if you want to cover a whole 3d-space, set '`receivers`' to '`auto`' and make sure that '`coverageMaxLevel`' is higher than '`coverageLevel`'
- if you want to receive the signal at specific points, you can either specify them under the configuration-key '`receivers`' (`receivers=[[0,0,1],[1,1,1],...]`) or in a file as command argument

Please check the sample files '`sample_lines.tr`' and '`sample_list.rec`' in the configfolder.

Please also check the '`sample.cfg`' in the configfolder for further customization.

4 Configuration keys

Available configuration keys for RaLaNS are listed with their default configuration value in the following table.

Key	Default value	Description
borders	`[]`	Usually the borders are determined from data, but you can limit the coverage area manually here. Format: [Left, Bottom, Right, Top]. For example: [-51, -27, 53, 21]
buildingHeight	10	sets the height of each building (used for OSM-maps); osm maps usually do not contains heights of buildings, set a overall building height
calculationsServers	`()`	you can distribute calculations to other computers, which are running calculationServer.py (only used if you have more transmitters than 'numberWorkers'), ('ip1','ip2',...) Please read section 2.1.6 and section 3.2 before usage.
center	`None`	Usually the center is determined from data and used as origin, but you can choose a center manually in UTM (Universal Transverse Mercator) format. Format: [Easting, Northing, 0.0]. sedanplatz example: [433663.812,5793036.36,0.0]
cluster	`False`	set to `True` for internal clusterUsage, You can edit the default settings above or set commands for all three jobs in the following lines | it is recommended to run the script with only -m as param before and set the generated zipfile as inputfile
clusterArray	`None`	set ArrayIndices to calculated transmitter with ids in the given range, format: X_Y | X,Y start and end indices
clusterCores	`None`	has to be this syntax: select=X:ncpus=Y | where X is the amount of nodes and Y is the amount of cores. Usally you dont have to change the default settings: select=1:ncpus=1
clusterLimit	`None`	set the amount of parallel calculation at once on cluster, only working on inf-cluster
clusterMail	`None`	your email-address
clusterMem	`None`	set the amount of memory your scripts use
clusterMsg	`None`	'bea' for messages if execution **[b]**egins, **[e]**nds and **[a]**borts, you don't have to write all 3 letters
clusterName	`None`	the name which will be shown by qstat, _ is not allowed on hpc2
clusterType	0	0: inf-cluster with Torque-System 1: hpc2 with PBSPro

Key	Default value	Description
clusterWtime	`None`	time after your program will be aborted, defaults: (hpc2: 48h), (inf: 168h)
coverageLevel	1	height at which 2d coverage calculation will be performed
coverageMaxLevel	1	3d calculation will be performed between coverageLevel and coverageMaxLevel
deadDistance	0.5	minimal distance between events
debugLevel	2	1: just config and outputfile 2: adds files for 2D-Viewer 3: adds files for 3D-Viewer 4: complete tmp-folder 5: adds debug-messages
debugRays	`False`	set `True` to draw rays in 3d-Viewer
diffractionThreshold	0.125*7	influence range of edges, $7 \times \lambda$, λ = wavelength, [3]
groundHeight	0	sets height of ground (used for OSM-maps)
interference	`False`	enable (`True`)/ disable(`False`) multi-path effects
mapName	`None`	Don't edit this. It will be set from the script.
maxIterations	50	force stop when a ray is reflected very often
name	'auto'	sets name of outputfile: <mapname>_name. Set value to 'auto' if you just want a timestamp (%Y%m%d-%H%M%S) after mapname
maxRange	0	don't calculate signal strength if distance between receiver and transmitter is too far, will speed up calculation for larger maps
numberWorkers	4	number of workers, only coverage maps are calculated in parallel, should be the number of cpu-cores also see section 3.2
port	`None`	not used, default port is 4242, you can change it in src/util/networking.py
ppServers	`()`	specify tuple of ppServers for cluster computing (doesn't work, although it should -> incompatible versions? network configuration? problem finding raylauncher? unfortunately no useful error message) use param calculationServers if you want parallel caluclation (replaces ppServers with a self-built solution)
preprop	`True`	set this to `False` if you already done the preparation for a map, make sure parameter 'name' is the same
rayNumber	5000	number of rays launched at sources
receivers	'auto'	set specific receiver here: if you want to calculate just some links: [[0,0,1],[1,1,1],...], for full coverage: 'auto' or 'full' for street coverage: 'street'
receiverType	0	Don't change manually! It is set by the script, default value: 0

Key	Default value	Description
receiveThreshold	'full'	size of "antenna", set to full and it will be calculated by script to $sqrt(3 \cdot (ss^2))/2$, should be half of `stepSize`
reflectionPart	0.153	sets what part of the incoming energy is reflected, value according to [2]
scatteringPart	0.0181	sets what part of the incoming energy is scattered, value according to [2]
stepSize	1	discretization of the coverage maps, receiver grid distance
terrainHeight	2	resolution of terrain #ray-caster
terrainLevel	'auto'	make a flat terrain instead of analyzing building heights
terrainWidth	2	resolution of terrain
timeout	0	timeout in seconds of a single job: one job is one receiver <-> transmitter link
transmitters	[[0, 0, 1]]	coverage maps will be calculated for each position, use 'area' or 'cubic' for full coverage or 'street' for street coverage
transmitterType	0	Don't change this value manually! It is set by the script, default value: 0
transmitterHeight	1	used if transmitter are placed along streets
wavelength	3e8 / 2.4e9	3e8 / 2.4e9 = 0.125 is the wavelength [speed of light / frequency] for 2.4 GHz antennas

Note: If `coverageLevel` = `coverageMaxLevel` it represents the `receiverHeight`.

5 Format of Result file: result.txt

The file generally consists of two parts: the header and the signal strengths generated by *RaLaNS*. The exact format of the file is determined with regard to the transmitter and receiver types.

Header

In general the header comprises of two or three lines. The first line gives information on the transmitters, the second line on the receivers. The first number in each of these lines depicts the Simulation Types. The line continues with information according to the given simulation type.

Signal strengths

After the header all calculated signal strengths are listed regarding the types of the transmitters and receivers. The signal strengths are displayed as decibel values in *scientific notation*[1]. If the strength is less than `-1e+03`, it will not be displayed in a corresponding plot. If the strength equals `-3.076526555685887615e+03` in 'result.txt', there exists no signal between transmitter and receiver. This value is the decibel value of the minimum positive floating point value, which can be calculated using *python* with the following statement:

```
"\%.18E" \% decimal.Decimal(10.0*numpy.log10(sys.float\_info.min))
```

Some formatting examples are given below. The following acronyms are used:

- Trans = Transmitter
- Recv = Receiver
- left, bottom, right, top → bounding box
- sSize = stepSize
- NorthingOffset : UTM Northing offset to center of map
- EastingOffset : UTM Easting offset to center of map

Point to Point

```
transType(0) TransX TransY TransZ
recvType(0) RecvX RecvY RecvZ
signal strengths from receivers to transmitter
```

Point to Area

```
transType(0) TransX TransY TransZ
recvType(2) left bottom right top coverage(Max)Level sSize sSize
signal strengths of all receivers in area to transmitter
```

[1] `https://en.wikipedia.org/w/index.php?title=Scientific_notation&oldid=731282070#E_notation`

Point to Cubic

```
transType(0) TransX TransY TransZ
recvType(3) left bottom coverageLevel right top coverageMaxLevel sSize sSize
   sSize
signal strengths of all receivers in area to transmitter
```

Point to List

```
transType(0) TransX TransY TransZ
recvType(4) #receiver <NorthingOffset EastingOffset coverage(Max)Level>
left bottom right top
signal strengths from receivers to transmitter
```

List to List

```
transType(4) #transmitter <NorthingOffset EastingOffset transmitterHeight>
recvType(4) #receiver <NorthingOffset EastingOffset coverageLevel>
left bottom right top
for each transmitter a row, each column represents a receiver
```

6 Additional informations on cluster usage

There must be at least two transmitters for cluster calculation. If speed from cluster with only one transmitter in a scenario is needed, randomly set a second transmitter. Calculation will be equivalent fast as with only one transmitter.

Every transmitter in a scenario will generate a process on the cluster.

7 Examples

Basic Example

Computing signal strengths:

```
python run.py inputfiles/small_street_flat.osm name='example'
```

Get a 2D result view:

```
python viewer2d.py outputfiles/small_street_flat_example.zip
```

Advanced Debugging Example

You can get a 3D debugging view with:

```
python run.py inputfiles/small_street_flat.osm name='debug' debugLevel=5
   debugRays=True
python viewer3d.py outputfiles/small_street_flat_debug.zip [0,0,1]
```

Preparing a file for *ns-3*

For usage in *ns-3*, you will usually need to calculate several virtual transmitter positions in order to place devices arbitrarily later on in *ns-3*. This calculation will last a bit longer (2 minutes on an an Intel i7):

```
python run.py inputfiles/small_street_flat.osm name='cover' stepSize=5
   transmitters='area'
```

Visualize signal strength distributions for different transmitter postions:

```
python viewer2d.py outputfiles/small_street_flat_cover.zip [3,2,1]
python viewer2d.py outputfiles/small_street_flat_cover.zip [53,2,1]
```

Scaling

When playing with the size of the map, the *stepSize* and the *rayNumber*, you will soon run into long calculation times. For a slight speedup you can limit the ray launching to a certain area around the transmitter and to positions on streets (the next step will be approx. 10 minutes on an Intel i7):

```
python run.py inputfiles/sedanplatz.osm name='streets' maxRange=350 stepSize=5
    transmitters=[[0,0,1]] receivers='street' rayNumber=100000
```

Plotting is slightly different, too:

```
python viewerScattered.py outputfiles/sedanplatz\_streets.zip -b
```

8 Installation guide for *ns-3* integration

This installation was done on Ubuntu 14.04.

Before downloading and installing *ns-3* the usual way, do

```
sudo apt-get install python-dev python-pygccxml python-numpy python-matplotlib
```

For using the calculation for lists, you have to download `ANN`:

```
sudo apt-get install libann0 libann-dev
```

Then download and install *ns-3* the usual way (see tutorial). For example look at this tutorial: `https://www.nsnam.org/docs/tutorial/html/getting-started.html`

Copy files from this order into the order containing the `waf` script (`ns-3-dev/` in older versions, simply something like `ns-3.21/` in newer versions). More explanation to the copy process:

- You need to copy all `C++` files, so also the `RaLaNSALone` and `RaLaNSList`, because the `NS3RaLaNS` is based on that one.
- So if you are in a folder similar to `ns-3.21`, you have to put all the `C++` files in the folder `src/propagation/model/`.
- Then you also need to update the `wscript`, either by just replacing it in `src/propagation` or integrating the necessary content into the existing file. If you have not added any new modules to that `wscript`, then replacing is the easier option. If you have already added new stuff to that `wscript`, then you can just add the lines, which have a comment with the word necessary.

Now you are good to go:

```
python ralanstest.py -m coverage/small_street_flat_example.zip -s 1 -d 1
```

If you need information about how to use `ralanstest.py`, just type:

```
python ralanstest.py -h
```

If you want to set some default values for the `ralanstest` script, then you an just open it and change the constants at the beginning of the file with valid values.

For your own simulations, you can use a result from the raylauncher (in `ralans/outputfiles`) just like the examplefile `coverage/small_street_example.zip`. Look at `ralanstest.py` and `scratch/ralans_example.py` for a usage example.

8.1 Examples

The following examples should give you an idea of what you are able to do with this software. The input files are the same ones as the ones, that you might have generated on your own using the examples for *RaLaNS* itself.

Simple example from above

```
python ralanstest.py -m coverage/small_street_flat_example.zip -s 1 -d 1
```

Interpolation

The ns-3 RaLaNS extension offers interpolation between values, so you can use smaller step sizes than in the input file.

```
python ralanstest.py -m coverage/small_street_flat_example.zip -s 0.3 -d 1
```

Transceivers

Usually you will want to use an inputfile with multiple virtual transmitters and receivers, so you can place transceivers wherever you want.

```
python ralanstest.py -m coverage/small_street_flat_cover.zip -s 1 -d 5 -t
   [-20,0,1]
python ralanstest.py -m coverage/small_street_flat_cover.zip -s 1 -d 5 -t
   [20,0,1]
```

Streets

You can use a file with transmitters and receivers only on streets as well (ignore the messages about receivers far away - this comes from only having receivers along the streets).

```
python ralanstest.py -m coverage/sedanplatz\_streets.zip -s 5 -d 5 -b
   [-472,-407,479,515]
```

8.2 Troubleshooting

ns.applications not found python bindings not enabled in ns3.24 (fixed in ns3.24.1):

see: `https://www.nsnam.org/wiki/Ns-3.24-errata`

or short, type:

```
./waf configure --enable-examples --enable-tests --with-pybindgen=../pybindgen
   -0.17.0.post41+ngd10fa60
```

References

[1] T. Hänel, A. Bothe, and N. Aschenbruck. "RaLaNS: A ray launching based propagation loss model for ns-3". In: *Proceedings of the International Conference and Workshops on Networked Systems (NetSys)*. 2015, pp. 1–7. DOI: 10.1109/NetSys.2015.7089069.

[2] O. Landron, M. J. Feuerstein, and T. S. Rappaport. "In situ microwave reflection coefficient measurements for smooth and rough exterior wall surfaces". In: *Proceedings of the 43rd IEEE Vehicular Technology Conference*. 1993, pp. 77–80. DOI: 10.1109/VETEC.1993.510972.

[3] U. M. Stephenson and U. P. Svensson. "An improved energetic approach to diffraction based on the uncertainty principle". In: *Proceedings of the 19th Int. Congress on Acoustics (ICA '07)*. 2007. URL: http://akutek.info/Papers/US_Uncertainty_Principle.pdf.