Urban Radio Attenuation Emulator

Craig Cooper*, Abhinay Mukunthan*
*ICT Research Institute - University of Wollongong, Wollongong NSW 2500, Australia
email: andor734@gmail.com

***Abstract*—This is the help document on URAE.**
***Index Terms*—Channel Modelling, Help**

I. Introduction

This is the help document for the Urban Radio Attenuation Emulator (URAE), developed by Craig Cooper and Abhinay Mukunthan. It has been released under the GLGPL.

This documents contains instructions on how to use the model in VEINS [1], a simulation framework that combines OMNeT++ [2] with SUMO [3] via the TraCI communications protocol [4]. The modules are built upon MiXiM [5]. To build URAE, VEINS 2.0 or later is required.

The model combines calculations to account for three separate elements of the channel:

1) Path Loss - Accounted for using the CORNER model proposed by Giordano et al [6]. This uses the layout of an urban environment to determine whether a destination node is within Line-Of-Sight of a transmitting vehicle. Different path loss calculations are applied based on this classification. The model was verified and improved upon by Mukunthan el al [7].
2) Fading - An environmentally dependant fading model was presented in [8], which uses the building layout to estimate the multi-path components and estimate the parameters of a Ricean fading model. A pre-simulation step to compute the fading parameters is required.
3) Vehicular obstructions - A deterministic model based on Knife-Edge shadowing was developed and verified experimentally by Wang et al [9]. This was selected to account for the transient nature of vehicular obstructions.

II. Setup

A. Building the library

To build URAE for OMNeT++, open a terminal and navigate to where you unzipped the URAE folder. Type the following command:

```
make install_OMNETPP
 OMNETPP_INSTALL_DIR=<install_dir>
 VEINS_ROOT=<veins_dir>
```

This will compile the URAE library, and copy the headers and library files to the given installation directory. The directories provided must be relative directories.

If you want to build in debug mode, use

```
make install_OMNETPP DEBUGMODE=1
 OMNETPP_INSTALL_DIR=<install_dir>
 VEINS_ROOT=<veins_dir>
```

Note, you should make sure you've built VEINS before building URAE, otherwise you'll get some annoying errors about cpp and cc files in OMNeT++.

B. Building the utilities

URAE has several utilities for preparing data for the engine. Most of these are written in Python (See Section III-A), but two are written in C++ and need to be built:

1) *BuildingSolver* - a tool that constructs an outline of buildings from a SUMO map.
2) *Raytracer* - a tool that performs the K-factor approximation method in [8] to a given SUMO map.

To build these, type:

```
make BuildingSolver
```

or

```
make Raytracer
```

The binary will be stored in the *bin* directory of the URAE root folder.

III. Utilities

URAE has a number of utilities that are used to prepare road data for simulations.

A. Sumo2Corner

Our CORNER implementation is different from that of UCLA [6]. Their approach used geometric distance calculations to determine the link on which a transmitter and receiver were travelling, and then classified the Line-Of-Sight (LOS) on the fly. This method is not computationally efficient for very large simulations, unfortunately. We have leveraged the unique features of VEINS, which allows us to simply fetch the current road IDs from SUMO. The LOS state can then be indexed in a database of link-pairs and corresponding classifications. By matching the textual Road ID to a numerical ID, the model is made more efficient. This necessitates a presimulation step of analysing the road network and analysing the classifications between all links in the network.

The script *Sumo2Corner.py* contains all the preprocessing functions for preparing a map to work with the CORNER components of URAE. It can operate on either a SUMO network file, or an OSM file.

The script accepts a valid filename, analyses the road network, and generates five files:

1) Link Lookup - A file of road links and numerical IDs of their corresponding nodes. Number of lanes is also included here.
2) Node Lookup - A file of road intersections and their geographical locations within the map.
3) Classification database - This database contains a list of source-destination road links, their LOS state (LOS, NLOS1, or NLOS2), and additional street information required for the pathloss calculation.
4) Link-Name Lookup - The classification database stores links and nodes as numerical indices for performance consideration. However, certain functions like the K-factor lookup relies on knowing the ASCII name of the link the current vehicle is occupying, as well as its lane. Thus, this lookup table allows URAE to find the ID of a link based on its name.
5) Internal Link Lookup - SUMO includes internal links, which connect lanes across intersections. If a map is created with internal links, an additional lookup is required to match an internal link to the intersection the car is crossing. This will allow the CORNER functionality to determine an LOS state for cars crossing intersections.

Sumo2Corner.py may also work on OSM maps. When given an OSM map, it extracts the road geometry into a SUMO net file, while also extracting the building geometry data for use with the Raytracer (See Section III-C). The script may be given the option of stripping out paths such as walkways and bicycle tracks, which the SUMO converter mistakes for vehicular roads.

To use with a SUMO network file, use the command:

```
Sumo2Corner.py -n <netfile>
```

This will use the base filename, sans extension, to name the files it generates.

To use with an OSM map, use the command:

```
Sumo2Corner.py -o <osmfile>
```

To strip walkways from the OSM file before conversion, use the **-s** option.

Note, if you specify your own SUMO map, you won't be able to get building geometry using this script. To generate building geometry, use the *BuildingSolver* tool (See Section III-B).

B. BuildingSolver

Sumo2Corner.py can generate building geometry data from an OSM map, but not from a SUMO network file. The *BuildingSolver* tool traces around the road network and generates building outlines in between roads. This data can then be fed into the Raytracer (See Section III-C) for K-factor calculation.

To use program, type the following command:

```
BuildingSolver <basefile>
  <lanewidth> <footPathWidth>
```

This script will load the CORNER files generated in the previous section (specified by **basefile**) and generate a set of buildings. The filename will be **basefile.corner.bld**.

C. Raytracer

The Raytracer is significantly more complicated than the others. As this program can be computationally intensive, we have attempted to write it in a manner conducive to parallel processing. To that end, the program first generates configurations, dividing the map area into several smaller segments. This allows multiple instances of the program to be run on different areas of the map simultaneously.

Note: This technically isn't a raytracer, but a ray-launcher.

To begin, run *Raytracer* with the **-g** option, followed by these configuration tags:

- **-b** - Specify base filename of CORNER files as generated by *Sumo2Corner*. Manditory.
- **-r** - Specify a ray count. Default: 256
- **-G** - Receiver Gain. Default: 1
- **-i** - This is the number of metres between consecutive positions for K-factor approximation. Default: 10
- **-c** - Number of cores to use in tracing the rays. Default: 2
- **-N** - Number of areas to divide the map into. Default: 1
- **-l** - Road width in metres. Default: 5
- **-F** - Filename to write configurations into. Default: config
- **-V** - Visualise the raytracing in progress. See Section III-C1

Once the configurations have been generated, run the program like so:

```
Raytracer <config> <run_number>
```

If you have a script or system that allows you to run a program with multiple configurations (such as a simulation cluster program), you can adapt your system to distribute *Raytracer* over multiple computers, to be run in parallel. When the entire run is complete, you will be presented with a file named **basename-run#.urae.k**. The basename will be the one specified in the configuration. These files will need to be combined into one file. At present, such a functionality has yet to be programmed.

Each file begins with the increment value specified in the file, followed by the number of source links contained within. The data is then organised in order:

1) Source Link ID
2) Increment along Source Link
3) Source Lane
4) Destination Link ID
5) Increment along Destination Link
6) Destination Lane
7) K-factor

When combining, ensure that the database is sorted according to Source Link ID.

1) Visualiser: It is possible to visualise the Raytracer program in progress. The program must be built for this first, using the command:

```
make Raytracer USE_VISUALISER=1
```

This uses Allegro 5.0 [10], a cross-platform multimedia library, to display the raytracer functioning. In order to build the visualiser, you will need a working installation of this library.

D. vehicleTypes.py

The script *vehicleTypes.py* allows you to assign specified vehicle types to cars in an existing SUMO route definition file. The script is executed with the command:

```
vehicleTypes.py <car_def> <in> <out>
```

The car definition file is a csv file. Each line has the format:

1) **id** - Name of the car type.
2) **accel** - Acceleration coefficient.
3) **decel** - Deceleration coefficient.
4) **sigma** - Driver imperfection.
5) **color** - Colour in SUMO GUI interface.
6) **length** - Length of car.
7) **width** - Width of car.
8) **height** - Height of car.
9) **weight** - Probability of this car being selected at random $(0, 1)$.

The first six parameters are explained in more detail on the SUMO wiki. The dimensions of the car are used in the shadowing model. Note that the total weight of all the cars in the file must sum to 1. An example vehicle definition file is in *data/vTypes.csv*.

IV. Running Simulations

A. Incorporating the model

The simplest way to start simulations using URAE is to subclass the *UraeScenarioManager* class. This object, when initialised, loads your CORNER and URAE files into memory automatically. Then you must create your vehicles in OMNeT++, specifying *UraePhyLayer*, or a subclass thereof, as your PHY layer module. Then you can specify which parts of the model in the analogue models XML file. This is an example of the analogue models section of the channel configuration file.

```
<AnalogueModels>
    <AnalogueModel type="CORNER">
        <parameter name="interval"
                   type="double"
                   value="0.1"/>
     </AnalogueModel>
    <AnalogueModel type="CarShadow">
        <parameter name="interval"
                   type="double"
                   value="0.1"/>
        <parameter name="wavelength"
                   type="double"
                   value="0.124378109"/>
    </AnalogueModel>
</AnalogueModels>
```

URAE is modular, meaning you can use only the components you wish. If you want only to use CORNER, you can omit the *CarShadow* segment. The environmental K-factors can be ignored in favour of a static value by adding the parameter *"k"* to the *CORNER* segment and setting a desired value. Conversely, if you want only the shadowing model, the *CORNER* segment can be removed. At present, the fading and CORNER functionality are inseparable.

B. Configuration

The *UraeScenarioManager* requires the following parameters to be specified.

- **linksFile** - Link file as created by *Sumo2Corner*.
- **nodesFile** - Node file as created by *Sumo2Corner*.
- **classFile** - CORNER Classification database.
- **linkMapFile** - Link name-ID lookup file.
- **intLinkMapFile** - Internal Link-node lookup file.
- **riceFile** - Precomputed K-factor database file, generated from *Raytracer*.
- **laneWidth** - Width of the lanes. This must be the same as was specified to the pre-simulation programs.
- **txPower** - Transmission power in milliwatts. This should be the same as was specified to the *Raytracer*.
- **systemLoss** - A loss factor associated with thermal noise. The value used in [7], [8] was 1142.9.
- **sensitivity** - Receiver sensitivity.
- **lossPerReflection** - Loss per reflection. This is not the value used in the *Raytracer*, but is used in the CORNER calculation.
- **componentFile** - Name of a datafile containing uncorrelated fading waveforms. This is used in calculation of the fading gain. The data used in [7], [8] is in *data/default.fading*.

V. Open Problems

The following problems need to be addressed.

A. BuildingSolver

The *BuildingSolver* has problems when constructing buildings between roads that meet on angles other than 90 degrees. Additionally, the building material is preprogrammed as brick. The algorithm needs to be fixed so that buildings can be constructed around roads meeting at any angle, and different building materials need to be specified. Ideally, this program should be built into the *Sumo2Corner* script.

B. Raytracer

This program requires support for scattering from other stationary elements of the environment, such as trees and poles. Additionally, RSU support should also be implemented.

VI. Conclusion

Bug reports or changes made, email us!

References

[1] C. Sommer, “Veins (Vehicles in network simulation),” 2010. [Online]. Available: http://veins.car2x.org/

[2] A. Varga, “OMNeT++,” 2009. [Online]. Available: http://www.omnetpp.org/

[3] D. Krajzewicz, E. Nicolay, and M. Behrisch, “SUMO - simulation of urban mobility,” 2011. [Online]. Available: http://sumo.sourceforge.net/

[4] A. Wegener, M. Piórkowski, M. Raya, H. Hellbruck, S. Fischer, and J. Hubaux, “TraCI: an interface for coupling road traffic and network simulators,” in *Proceedings of the 11th communications and networking simulation symposium on - CNS ’08*, Ottawa, Canada, 2008, p. 155.

[5] A. Viklund, “Mixim project,” 2011. [Online]. Available: http://mixim.sourceforge.net/

[6] E. Giordano, R. Frank, G. Pau, and M. Gerla, “CORNER: a radio propagation model for VANETs in urban scenarios,” *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1280–1294, Jul. 2011.

[7] A. Mukunthan, C. Cooper, D. Franklin, M. Abolhasan, F. Safaei, and M. Ros, “Experimental validation of the corner propagation model based on signal power measurements in a vehicular environment,” in *IEEE WCNC 2013*. Shanghai: IEEE WCNC, 2013.

[8] C. Cooper, A. Mukunthan, M. Ros, D. Franklin, and M. Abolhasan, “Dynamic environmental fading in urban vanets,” in *IEEE International Conference on Communications (ICC) 2014*, 2014.

[9] P.-J. Wang, C.-M. Li, and H.-J. Li, “Influence of the shadowing on the information transmission distance in inter-vehicle communications,” in *IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 2009, pp. 3015–3019.

[10] S. Hargraves, “Allegro 5.0,” 2014. [Online]. Available: http://alleg.sourceforge.net/