DRIVE: Digital Twin for self-dRiving Intelligent VEhicles

User Manual to Accompany DRIVE Simulation Tool

Ioannis Mavromatis
ioan.mavromatis@bristol.ac.uk
ioannis.mavromatis@toshiba-bril.com

Table of Contents

1. Introduction

This user manual describes the implementation of DRIVE, which is a "Simulation Framework for City-Scale Experimentation". This framework is designed in a way to reduce the computational complexity of the existing simulation frameworks, still providing a very realistic representation of the real-world. Some of its main features are:

- Supports different communication planes in a heterogeneous-fashion and provides an easy way of adding them and modifying their configuration during the simulation time.
- Highly vectorised and parallelised environment, minimising the execution time by utilising several optimisation capabilities from MATLAB.
- Imports buildings and foliage outlines from OpenStreetMap for a realistic city environment [1].
- Imports the positions of the street furniture as potential basestation positions around a city. The positions are imported either from OpenStreetMap [1] or from SUMO traffic generator [2].
- Provides a bidirectional connection with SUMO traffic generator [2], via Traffic Control Interface (TraCI) [3];
    - Easily imports vehicular and pedestrian mobility traces, generated via SUMO traffic generator.
    - Modify and manipulate the mobility traces on-the-fly based on decisions taken either by the end-user or by a decision-making algorithm.
- Provides a way of generating indoor users for all the given buildings on the map, that contribute to the overall network load.
- Visualisation of the mobility traces and the network performance on the imported city maps.
- Supports different large-scale signal variation models (path-loss and shadowing) for both LOS and NLOS scenarios; more models can be easily added in the framework.
- Easily extendable to support maps and vehicular traces from different sources.
- MATLAB implementation; Open-source code, free and openly distributed.

This user manual describes how to install and use DRIVE and gives a high-level overview of the developed functionalities. DRIVE can be used for investigating different communication-related problems, such as optimising the basestation placement and switch-on-switch-off approaches on the deployed basestations, heterogeneous resource allocation, efficient content dissemination and fetching, quality-of-service centric optimisation as perceived from the system-level perspective, etc. All the above problems can be tackled from both the indoor user, as well as the Vehicle-to-Everything (V2X) perspective. It is also possible to take both into account when considering the generated network load.

The framework is designed in such a way to be highly parallelised and vectorised. Therefore, the execution time is minimised, making it a great tool for developing AI- and Machine Learning algorithms for the above-mentioned problems. Of course, traditional optimisation algorithms can be used as well in a similar fashion. This framework, being designed in MATLAB, can be very easily linked with the existing optimisation and Machine-learning toolboxes provided by MathWorks. However, the MATLAB implementation does not limit the end-user to use machine-learning algorithms on different programming languages as well.

For example, this link describes how Python libraries can be called from within MATLAB. The framework is flexible enough to support such features.

2. Getting Started

2.1. Installation

The installation of DRIVE is straightforward. If the SUMO integration functionality is not required by the end-user, the steps are: download the code from the official repository, unzip it to a directory, and it is ready to be used! In a later section, we will describe the steps required for integrating the DRIVE framework with SUMO traffic generator as well. DRIVE is organised in "toolboxes" folders. A function adding all the required toolboxes in the MATLAB path is provided within the framework. This function is called during the initialisation time of the framework.

Some external libraries provided should be compiled as MEX files. At the moment these are:

- Lightspeed Matlab Toolbox: Installation instructions can be found under: https://github.com/tminka/lightspeed

DRIVE can work on any platform running a recent version of MATLAB. It has been tested with MATLAB 2019a and 2018b, on both MAC OS X and Ubuntu environments without any problems. It has also been tested with SUMO 1.2.0, but it is expected to run with any SUMO version >1.0.0.

2.2. Running the simulation

Starting from the root directory of DRIVE, in order to run the framework, the file *runSimulator.m* should be executed. This file will load all the required toolboxes in the MATLAB path (*setupSimulator.m*), load the initial configuration (*simSettings.m*), and execute the main body of the framework.

DRIVE provides two different functionalities regarding the loading/processing of the map (Fig. 1) and the mobility trace files.

- When the simulator runs for the first time with a given file as an input, the map and mobility trace files are parsed and imported in the framework. The end-users can decide whether they want to save the processed files or not.
- If the simulator finds an already pre-processed file in the *./mobilityFiles/preprocessedFiles/* directory, DRIVE asks the user whether these file will be loaded or not. If the user provides a positive answer, then the files are loaded from the pre-processed directory. On the other hand, the files are processed again and overwritten on the existing ones.

Figure 1 - An example of a map imported in DRIVE.

In Sec. 4, we will further describe the saving and loading functionalities of the simulator. After the parsing/loading of the map and the mobility traces is complete, the simulator calculates the LOS and NLOS links for the entire map. Again, as before, if a pre-processed file exists, the users can decide whether they are going to use that file or not. If a pre-processed file does not exist, everything is being calculated based on the given map and configuration.

2.3. Improving the execution time

During the development of DRIVE framework, particular attention was given on parallelising and vectorising several sections of the code to speed-up the execution time. The parallel execution of several functions can be enabled by changing *SIMULATOR.parallelRun* variable to one, found in the *simSettings.m* file. If the parallel execution is enabled, the end-user can later specify the number of cores to be allocated for the simulation, modifying *SIMULATOR.parallelWorkers*. This can significantly decrease the execution time, especially during the initialisation phase of the scenario where the map is imported, and the LOS/NLOS links are calculated.

2.4. Map Simplification to Decrease Complexity and Reduce the Execution Time

Several map simplification strategies take place within DRIVE. At first, the building and foliage polygons are merged, and their inner holes are removed. By that, we can achieve a very realistic performance investigation from the system-level perspective, minimising the computational complexity of the excessive number of polygons. This functionality is manipulated by the end-user modifying the *map.simplificationTolerance* parameter. Doing so, the polygon edges are smoothened removing the excessive corners, thus reducing the time needed to calculate the LOS and NLOS links for the entire city.

On top of that, a tessellation approach for the entire map is followed, in a two-way fashion. At first, the user can define the size of the map areas (*map.area* in the configuration file). This discretises the map into different large areas with equal size, that can be later used to either apply different policies or evaluate different parameters. The areas can have either a hexagonal or a squared shape (Fig. 2) that can be defined by the user modifying *map.areaShape* parameter (1 for hexagon and 2 for square). Also, the map is tessellated in tiles as well (*map.tileSize* parameter). A tile is considered as a unified area on the map that has exactly the same properties on its entire surface. For DRIVE, all the calculations take place using the incentres of the given tiles to speed up the performance. Using a relatively small tile

size (e.g., <5m), a very realistic result can be achieved significantly reducing the computational complexity. Again, more information about this approach can be found in [5].

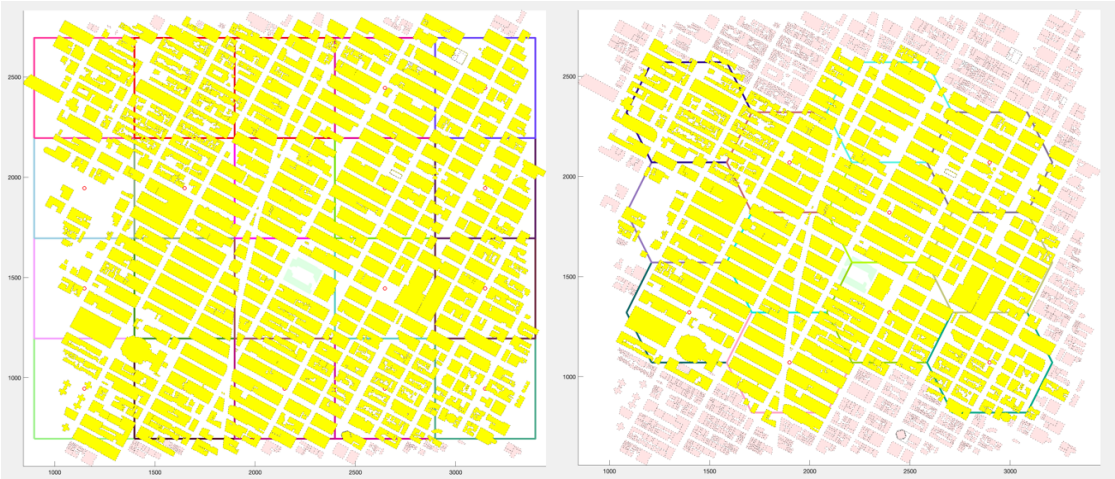

Figure 2 - Examples of squared and hexagonal areas on the city.

2.5. Generating Indoor User Traffic

DRIVE provides the functionality to introduce a number of users per building block. The maximum number of users for a given area is specified in the configuration file using the variable *map.maxUsersPerBuilding*. If zero is given, the indoor users will not be taken into account. On top of that, the user can define the granularity of the indoor model using *map.densitySimplification*. This variable defines the area of squared-tiles that the above-mentioned number of users will be applied. For example, if the maximum number of users is 50 and the density simplification parameter is set to 100, then each squared tile with surface area of $100m^2$ can have up to 50 users. Based on the number of users per surface area, the number of users per building block is later calculated. The user density model used in DRIVE is described in [4] and is based on real-world experiments.

2.6. Introducing new Communication Technologies

The different communication planes within the simulation framework are categorised either as macro-cell-based or femto-cell-based. The main difference between the two categories is the position of the basestations. For the macro-cell approach, the basestation is positioned on top of the buildings. More specifically, it is being placed at the incentre of a building block (described as a large polygon in the simulator). The femto-cell basestations are considered to be mounted on top of the street furniture. Importing the positions of the traffic lights from the map file, we create a set of potential positions for basestation deployment. OSM does not provide information about the lampposts. Therefore, in order to provide coverage for roads that don't have traffic lights, a road is discretised in equal sections and lamppost are considered to be equally spaced, based on a given distance threshold defined as *femtoDistanceThreshold*. More details about that can be found in [5].

The configuration files for each communication plane should be added within the *./ratToolbox/available/* directory. As many communication configuration files, as needed can be added by the end-user. Within DRIVE, we provide two examples of the two different categories, i.e., "lte" referring to the macro-cell one, and "mmWaves" referring to the femto-

cell approach. These files can act as templates for the end-user and can be modified accordingly to introduce the required functionality.

2.7. Simulation output

Different verbosity levels are provided by the simulation framework (0, 1, and 2). The different levels can be changed by modifying *VERBOSELEVEL* variable in the *simSettings.m* file. If the verbosity level chosen is greater or equal to 1, then the simulator plots several figures (Fig. 3) with the performance of each communication plane and their datarates. The verbosity level also dictates the output on the Command Window as well. For values greater or equal to 1, several text messages are printed showing the progress of the simulation and the time required for each step. The second verbose level plots several test figures as well, such as the different map areas and tiles. It can be used particularly when the end-user modifies the framework and extends its functionality to cross-validate several aspects of the designed functions. All the above can be disabled, by choosing 0 as the verbose level. – *This will be further extended to save the generated figures, before the official release of the framework.*

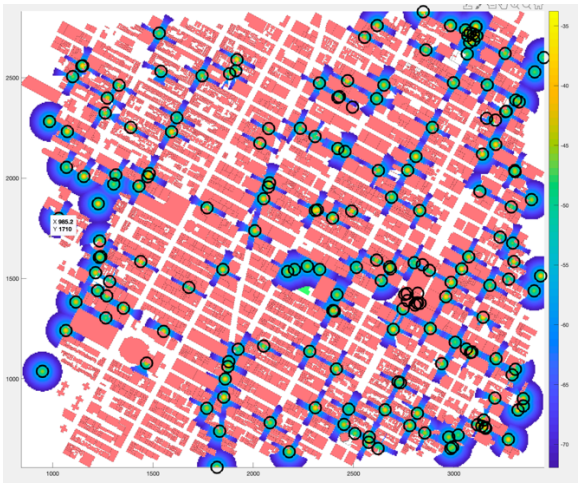

Figure 3 - A RSSI visualisation example for a given technology and a number of basestations.

2.8. Using SUMO and TraCI

The mobility traces used within DRIVE are generated using SUMO traffic generator. In order to utilise the SUMO mobility traces and maps, a link between SUMO and DRIVE should be established. For that, TraCI is used. TraCI allows the user to interact with SUMO, in a client-server fashion where MATLAB acts as the client, and SUMO as the server. In order to do so, TraCI4Matlab should be added in the MATLAB path at first and the additional Java dependencies to the MATLAB's static Java path. If everything works correctly, SUMO can be used to generate a scenario. Having the scenario, SUMO can be later executed in the server mode from MATLAB and initialise the connection between the two. TraCI commands can be later used to progress the scenario (e.g., progress to the next timestep), and parse the various information provided (e.g., get the position of all the vehicles for a given time). More information about how to setup and use TraCI can be found under the link.

3. DRIVE framework dependencies

DRIVE requires the following functions, that are publicly available either from MATLAB Central File Exchange, under https://uk.mathworks.com/matlabcentral/fileexchange/ or on GitHub repositories. These functions are copyrighted by their respective developers under a BSD licence. For convenience they are included in DRIVE framework, under *externalToolbox* directory, along with their corresponding licences.

- deg2utm © 2006 by Rafael Palacios: https://uk.mathworks.com/matlabcentral/fileexchange/10915-deg2utm.
- Line Simplification © 2010 by Wolfgang Schwanghart: https://uk.mathworks.com/matlabcentral/fileexchange/21132-line-simplification.
- Fast Line Segment Intersection © 2010 by U. Murat Erdem: https://uk.mathworks.com/matlabcentral/fileexchange/27205-fast-line-segment-intersection.
- INPOLY: A fast points-in-polygon test © 2018 by Darren Engwirda: https://uk.mathworks.com/matlabcentral/fileexchange/10391-inpoly-a-fast-points-in-polygon-test.
- MatGeom: Matlab Geometry Toolbox for 2D/3D Geometric Computing © 2019 by David Legland: https://github.com/mattools/matGeom.
- OpenStreetMap Functions © 2016 by Ioannis Filippidis: https://uk.mathworks.com/matlabcentral/fileexchange/35819-openstreetmap-functions.
- Lightspeed Matlab Toolbox © 2019 by Tom Minka: https://github.com/tminka/lightspeed
- xml2struct © 2012 by Wouter Falkena: https://uk.mathworks.com/matlabcentral/fileexchange/28518-xml2struct.
- Waitbar for Parfor © by Yun Pu: https://uk.mathworks.com/matlabcentral/fileexchange/71083-waitbar-for-parfor
- TraCI4Matlab © 2019 by Andres Acosta: https://uk.mathworks.com/matlabcentral/fileexchange/44805-traci4matlab.
- questdlg timer © 2011 by Az Nephi: https://uk.mathworks.com/matlabcentral/fileexchange/32977-questdlg-timer

4. Buildings and Foliage Polygons

A city map can be downloaded from OpenStreetMap (OSM). OSM is an open-source project that provides collaborative and editable maps from around the world. The OSM maps are generated and updated by volunteering end-users, performing systematic ground surveys using tools such as handheld GPS units, notebooks, digital cameras, or voice recorders. Having that in mind, the actual representation of buildings and foliage in an OSM map might be limited, having several missing objects. That said, DRIVE's end-users should be careful with the maps of their choice, as they may lead to unrealistic simulation outputs due to the missing objects.

OSM maps can be either directly imported into DRIVE framework or can be used to generate SUMO scenarios. For the first case, OpenStreetMap Functions package, available under this

link, can be used. A version has been included in our framework and modified accordingly to support different types of buildings and foliage polygons.

In order to use the exported OSM map, the user should specify that in the *simSettings.m* configuration file. At first, *SIMULATOR.map* should be set to zero, to specify that an OSM map should be used. Later, the variable *map.file* can define the file path and name to be used. The OSM file will be parsed and processed during the first execution of the simulation framework and it will be saved for future experimentation. All the pre-processed files are saved under the ./mobilityFiles/preprocessedFiles/osm/ directory. A folder with the map name is created and the processed map is saved in there. If the end-user decided to process again the map, the pre-processed file will be overwritten.

5. Using SUMO to generate Maps and Mobility Traces

A second option for DRIVE is to rely on maps converted already using SUMO *netconvert* command and importing the mobility traces for several vehicle types and pedestrians. An example of how to convert an OSM map to its SUMO equivalent and generate some mobility traces is as follows:

- netconvert --osm-files map.osm.xml --output-file map.net.xml --no-turnarounds --osm.elevation --sidewalks.guess --crossings.guess --remove-edges.by-vclass rail,rail_electric,bicycle,tram,rail_urban,bus --junctions.join --lefthand
- polyconvert --osm-files map.osm.xml --net-file map.net.xml -o map.poly.xml
- randomTrips.py --net-file=map.net.xml --additional-files=additional.type.xml --output-trip-file=map.trips.xml --route-file=map.rou.xml --begin=0 --end=2000 --verbose --trip-attributes="type=\"typedist\" departLane=\"best\" departSpeed=\"max\" departPos=\"random\""
- randomTrips.py --net-file=map.net.xml --additional-files=additional.type.xml --output-trip-file=mapPedestrian.trips.xml --route-file=mapPedestrian.rou.xml --begin=0 --end=2000 --verbose --pedestrians

The above example shows how to take a map from OSM, called “map.osm.xml”, convert it to a SUMO network file, generate the several building and foliage polygons later and finally, generate the vehicle and pedestrian mobility traces. More information about the usage and fine-tuning of the above commands can be found in their corresponding documentation pages, i.e., netconvert, polyconvert, randomTrips. Finally, more information about SUMO traffic generator and its functionalities can be found under the official documentation page under this link.

To use this map *SIMULATOR.map* variable should be set to 1. Again, the *map.file* points to the correct map file and folder. Relying on TraCI, we can import and manipulate the different polygons and node positions on the map, as it is described in TraCI’s documentation. To reduce the processing time, again the SUMO map is parsed only once, and it is later saved for faster loading time in the future. For the vehicular and pedestrian mobility, on each timestep SUMO is queried via TraCI’s interface and the positions of all the nodes are parsed and manipulated for the given time.

6. Referencing DRIVE simulation framework

TO BE ADDED LATER.

7. Acknowledgements

This work is funded in part by Toshiba Research Europe Ltd., in part by the Next-Generation Converged Digital Infrastructures (NG-CDI) project, supported by BT Group and EPSRC (EP/R004935/1), and in part by the CAVShield project (Innovate UK, no. 133898).

8. References

1) M. Haklay, and P. Weber, “OpenStreetMap: User-Generated Street Maps”, *in IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Dec. 2008.
2) P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic Traffic Simulation using SUMO", *in Proc. of IEEE Intelligent Transportation Systems Conference (ITSC) 2018*, Maui, Hawaii, United States, Nov. 2018.
3) Acosta, J. E. Espinosa, and J. Espinosa, “TraCI4Matlab: Enabling the Integration of the SUMO Road Traffic Simulator and Matlab Through a Software Re-Engineering Process”, *in Modeling Mobility with Open Data, Springer International Publishing*, pp. 155–170, 2015.
4) D. Lee, S. Zhou, X. Zhong, Z. Niu, X. Zhou and H. Zhang, "Spatial modeling of the traffic density in cellular networks," *in IEEE Wireless Communications*, vol. 21, no. 1, pp. 80-88, Feb. 2014.
5) I. Mavromatis, A. Tassi, R. J. Piechocki and A. Nix, "Efficient Millimeter-Wave Infrastructure Placement for City-Scale ITS," *in Proc. Of IEEE VTC-Spring 2019*, Kuala Lumpur, Malaysia, pp. 1-5, May 2019.