#Calibration of Microscopic and Mesoscopic SUMO Simulation Model based on the Real-world Speed Data

Calibration adjusts model parameters to improve the model's ability to reproduce time-dynamic system performance observed under specific travel conditions.

For the current project, the primary reference for the calibration is [Traffic Analysis Toolbox Volume III](https://ops.fhwa.dot.gov/publications/fhwahop18036/index.htm) Chapter five [Model Calibration](https://ops.fhwa.dot.gov/publications/fhwahop18036/chapter5.htm#calibrate-model-variant-to-meet-acceptability-criteria).

##Step 1. Modification of OD matrices based on the time-of-day

The matrices have used a set of [OD-matrices]() provided by [NREL](https://www.nrel.gov/). Based on the instruction provided, the proportion of each period of time is: 100% of peak hours plus 16.7% off-peak hours matrices. Due to a [bug](( https://github.com/eclipse/sumo/issues/8676) in [SUMO](https://www.eclipse.org/sumo/) to calculate 0 probability, if a timeline contains a time slice with 0 probability, vehicles may depart in that slice because fractional vehicles from a previous slice are emitted with some probability in a subsequent slice. Therefore, it is needed to calculate ODs manually with this time frame:

\*\*

0:0.167,21600:0,32400:0.167,54000:0,64800:0.167,86400:0

\*\*

##Step 2. Generate vehicle trips XML file

Use a SUMO tools [od2trips](https://sumo.dlr.de/docs/Demand/Importing\_O/D\_Matrices.html) to generate trips files for each vehicle type and time-of-day by incorporating transportation demand [in O format](https://sumo.dlr.de/docs/Demand/Importing\_O/D\_Matrices.html) and [taz.xml](https://github.com/smarttransit-ai/transit simulator/blob/master/manual\_files/SUMO\_simulation/taz.xml). Nine “trips.xml" file generate as below:

- trips\_pass\_am\_xml.xml

- trips\_pass\_pm\_xml.xml

- trips\_pass\_op\_xml.xml

- trips\_sut\_am\_xml.xml

- trips\_sut\_pm\_xml.xml

- trips\_sut\_op\_xml.xml

- trips\_mut\_am\_xml.xml

- trips\_mut\_pm\_xml.xml

- trips\_mut\_op\_xml.xml

Use python script [Combine\_trip\_files.ipynb]() to combine the xml file into a [combin\_trips.xml file]().

The command is shown below.

```

od2trips --taz-files taz.xml --od-matrix-files pass\_op.txt --output-file trips\_pass\_op\_xml.trips.xml --prefix pass\_op --vtype passenger --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files mut\_op.txt --output-file trips\_mut\_op\_xml.trips.xml --prefix mut\_op --vtype trailer --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files sut\_op.txt --output-file trips\_sut\_op\_xml.trips.xml --prefix sut\_op --vtype truck --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files pass\_am.txt --output-file trips\_pass\_am\_xml.trips.xml --prefix pass\_am--vtype passenger --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files mut\_am.txt --output-file trips\_mut\_am\_xml.trips.xml --prefix mut\_am --vtype trailer --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files sut\_am.txt --output-file trips\_sut\_am\_xml.trips.xml --prefix sut\_am --vtype truck --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files pass\_pm.txt --output-file trips\_pass\_pm\_xml.trips.xml --prefix pass\_pm--vtype passenger --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files mut\_am.txt --output-file trips\_mut\_pm\_xml.trips.xml --prefix mut\_pm --vtype trailer --spread.uniform t

od2trips --taz-files taz.xml --od-matrix-files sut\_pm.txt --output-file trips\_sut\_pm\_xml.trips.xml --prefix sut\_pm --vtype truck --spread.uniform t

```

##Step 3. Download network from Open Street Map (OSM)

Use SUMO tools [OSMWebWizard](https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html) to import a large-scale network. [OpenStreetMap]( http://www.openstreetmap.org) is a free editable map of the whole world. It is made by people like you.".

\*\*Caution:\*\*

If the map excerpt covers a very large area, the simulation might become slow or even unresponsive.

##Step 4. Edit network

Use SUMO tools, [netconvert](https://sumo.dlr.de/docs/netconvert.html) to modify and edit the imported map from [OSM]( http://www.openstreetmap.org).

Several aspects of the imported network may have to be modified to suit your needs. Some of the relevant [netconvert] options are described below.

\*\*Option:\*\*

--geometry.remove : Simplifies the network (saving space) without changing topology.

--ramps.guess : Acceleration/Deceleration lanes are often not included in OSM data. This option identifies likely roads that have these additional lanes and causes them to be added.

--junctions.join: applies a heuristic to join these junction clusters automatically and is used by default when using a python script [osmBuild.py](https://github.com/eclipse/sumo/blob/master/tools/osmBuild.py)

--tls.guess-signals --tls.discard-simple --tls.join : See [traffic light](https://sumo.dlr.de/docs/Networks/Import/OpenStreetMap.html#traffic\_lights)

--tls.default-type actuated: Default static traffic lights are defined without knowledge about traffic patterns and may work badly in high traffic.

The command is shown as below.

```

--geometry.remove --ramps.guess --junctions.join --tls.guess-signals --tls.discard-simple --tls.join --tls.default-type actuated

```

Also, it is possible to build SUMO networks for very large areas (the whole of Chattanooga), but some precautions should be taken: [OSM for large-scale network](https://sumo.dlr.de/docs/Networks/Import/OpenStreetMap.html).

The following options can be used to reduce the network size:

\*\*Options:\*\*

--no-internal-links, --keep-edges.by-vclass passenger, --remove-edges.by-type highway.track,highway.services,highway.unsurfaced

if only major roads are needed, reduction of the network can be done by setting the option

--keep-edges.by-type highway.motorway,highway.motorway\_link,highway.trunk,highway.trunk\_link,highway.primary,highway.primary\_link

##Step 5. Traffic light modification

Use an iteration way between load network by [SUMO](https://sumo.dlr.de/docs/index.html) to show the name of the intersection with error on the traffic light, then edit [traffic light](https://sumo.dlr.de/docs/Simulation/Traffic\_Lights.html). Also, [sumo/sumo-gui]( https://sumo.dlr.de/docs/sumo-gui.html) allows loading definitions which describe when and how a set of traffic lights can switch from one program to another.

##Step 6. Convert TAZ shapefiles to polygon and polygon to edges.

Use [polyconvert]( https://sumo.dlr.de/docs/polyconvert.html) to import geometrical shapes [TAZ\_2014](…..) , converts them to a representation that may be visualized using [sumo-gui]( https://sumo.dlr.de/docs/index.html).

The command is shown as below.

```

polyconvert --net-file Chattanooga\_SUMO\_Network.net.xml --shapefile-prefixes TAZ\_TAZ\_2014 --shapefile.id-column id --shapefile.guess-projection t -o polygone.xml

```

##Step 7. convert polygon file to Sumo TAZ file with edges assigned to TAZs:

Use python script [edgesInDistricts.py]( https://sumo.dlr.de/docs/Tools/District.html). Parsing a number of networks and TAZ (district) files with shapes, this script writes a TAZ file with all the edges which are inside the relevant TAZ.

The command is shown below.

```

edgesInDistricts.py -n Chattanooga\_SUMO\_Network.net.xml -t o polygone.xml -o taz.xml

```

\*\*Option:\*\*

Below are additional options to determine edge processing:

edgesInDistricts.py –help

##Step 8. Generate trips file (Iterative Assignment (Dynamic User Equilibrium))

Use python script [duaIterate.py]( https://sumo.dlr.de/docs/Tools/Assign.html#dua-iteratepy) to perform the computation of a dynamic user assignment (DUA). It works by alternatingly running the simulation to discover travel times and then assigning alternative routes to some of the vehicles according to these travel times. This is repeated for a defined number of iteration steps. At least two files must be given as input the script: a SUMO network and a set of trip definitions. A stochastic user-equilibrium (UE) traffic state is not guaranteed after the assignment. Increasing the number of iteration steps increases the likelihood of convergence to equilibrium. Within each iteration step, the script generates a configuration file for the [duarouter]( https://sumo.dlr.de/docs/duarouter.html) application and starts it with this configuration file. Then, a configuration file for sumo is built, and sumo is started. Both configuration files are completely defined within the script itself.

The number of iterations may be set to a fixed number of determined dynamically depending on the used options. In order to ensure convergence, there are different methods employed to calculate the route choice probability from the route cost (so the vehicle does not always choose the "cheapest" route). In general, new routes will be added by the router to the route set of each vehicle in each iteration (at least if none of the present routes is the "cheapest") and may be chosen according to the route choice mechanisms like [Gawron](https://d-nb.info/1183255454/34) and [Logit]( https://www.tandfonline.com/doi/full/10.1080/03081061003643705).

The command is shown below.

```

duaIterate.py --net-file Chattanooga\_SUMO\_Network.net.xml -t combined\_trips.xml -l 2 -C -m --meso-recheck 1 -E -L --time-to-teleport 50 --clean-alt

```

Also, if not prefer using DUA, using a single assignment by [duarouter]( https://sumo.dlr.de/docs/duarouter.html) with the below command:

\*\*Options:\*\*

duarouter --route-files combined.trips.xml –net-file Chattanooga\_SUMO\_Network.net.xml --unsorted-input --additional-files busStopsCARTA.add.xml,vehtype.add.xml --ptline-routing --output-file Cattanooga.rou.xml --ignore-errors

However, only the shortest path were used instead of a user assignment algorithm may cause lots of jams/deadlocks.

##Step 9. Define some points on the network to accumulating speed data from [INRIX](https://inrix.com/products/speed/)

Use [Google Earth]( https://earth.google.com/web/) to determine some points ( points alongside the streets which is introduced by Geo-Location (Lat, Long) at the network to get the speed data from [INRIX]( https://inrix.com/products/speed/) speed data set.

##Step 10. Define induction loop detectors

The detectors are placed in various places as well as the original detectors in real world. [137 detectors]() were applied on the network to measure the speed and the flow. They are spread on the entry and exit edges of the networks. [E1 Loops Detectors]( https://sumo.dlr.de/docs/Simulation/Output/Induction\_Loops\_Detectors\_%28E1%29.html) are used and use python script [Def\_Detector\_File.py](https://sumo.dlr.de/docs/Tools/Detector.html) to generate [detectors.add.xml]() file by incorporating network file and edge converted Geo-location (Lat, long). An induction loop is defined this way within an [additional-file]( https://sumo.dlr.de/docs/sumo.html#format\_of\_additional\_files) like this:

```xml

<additional>

<inductionLoop id="<ID>" lane="<LANE\_ID>" pos=" <POSITION\_ON\_LANE>" freq="<AGGREGATION\_TIME>"

file="<OUTPUT\_FILE>" friendlyPos="true"/>

</additional>

```

Multiple definitions may be placed in the same additional-file and also reference the same output file. A single data line within the output of a simulated induction loop looks as following:

```xml

<xml version="1.0"?>

<interval begin="''<BEGIN\_TIME>''" end="''<END\_TIME>''" id="''<DETECTOR\_ID> \

nVehContrib="''<MEASURED\_VEHICLES>''" flow="''<FLOW>''" occupancy="''<OCCUPANCY>''" \

speed="''<MEAN\_SPEED>''" harmonicMeanSpeed="''<HARM\_MEAN\_SPEED>''" length="''<MEAN\_LENGTH>''" nVehEntered="''<ENTERED\_VEHICLES>''"/>

```

The detector computes the values by determining the times the vehicle enters and leaves the detector first. This implicates that a) some values are not available as long as the vehicle is on the detector, and b) some values cannot be computed if the vehicle enters the detector by a lane change - as the vehicle did not pass the detector completely.

##Step 11. Set parameters for the calibration (Microscopic and Mesoscopic)

The calibration is an important part of modeling since there is no model that can be accurate to the real traffic condition. The calibration is a kind of adaptation process for the model to represent the real traffic condition. Speed data is calibrated to get the optimal value of the model parameters by changing some parameters which predicted will directly influence the speed of the network. Every microsimulation software program comes with a set of user-adjustable parameters for the purpose of calibrating the model to local conditions. Therefore, the objective of calibration is to find the set of parameter values for the model that best reproduces observed measures of system performance [[1]](https://ops.fhwa.dot.gov/publications/fhwahop18036/chapter5.htm).

The traffic simulation tools can mainly be divided into four different groups [Stefan Krauß]( https://sumo.dlr.de/pdf/KraussDiss.pdf) :

1) Macroscopic: average vehicle dynamics like traffic density are simulated.

2) Microscopic: each vehicle and its dynamics are modeled individually.

3) Mesoscopic: a mixture of macroscopic and microscopic models.

4) Submicroscopic: each vehicle and also functions inside the vehicle are explicitly simulated e.g. gear shift.

### Step 11.1. Microscopic model

In the [SUMO]( https://sumo.dlr.de/docs/ ), the speed parameter is used in the calibration process. The reason was that the observation data only provide speed and flow. The flow data was not being used in the calibration process because the only data from the real world we have is speed data. Root Mean Square Error (RMSE) minimizing analysis and T-test were used to calibrate data between the model output and observed data to find the optimal value of parameters in the model development. In order to find the optimal value of parameters in the model, some parameters were changed or modified. The changed parameters in the calibration process are those in [Krauβ]( https://sumo.dlr.de/pdf/KraussDiss.pdf ) car-following parameters. After having several trial-and-error experiments, there are four parameters that occurred which can make significant changes in the network. Those parameters are sigma (driver imperfection), tau (driver reaction time), speedFactor, or speedDev.

\*Tau\* is intended to model a driver's desired time headway (in seconds). It is used by all models. Drivers attempt to maintain a minimum time gap of tau between the rear bumper of their leader and their own "front-bumper + minGap" to assure the possibility to brake in time when their leader starts braking, and they need tau seconds reaction time to start breaking as well.

\*Sigma\* is the driver imperfection (0 denotes perfect driving ).

Use python script [createVehTypeDistribution.py]( https://sumo.dlr.de/docs/Tools/Misc.html ) to generate a vehicle type distribution by sampling from configurable value distributions for the desired [vType-parameters](https://sumo.dlr.de/docs/Definition\_of\_Vehicles%2C\_Vehicle\_Types%2C\_and\_Routes.html).

The only required parameter is the configuration file in the format shown below (example config.txt):

```txt

tau; normal(0.8,0.1)

sigma; normal(0.5,0.2)

length; normal(4.9,0.2); [3.5,5.5]

param; myCustomParameter; normal(5, 2); [0, 12]

vClass; passenger

carFollowModel; Krauss

```

\*\*Options:\*\*

Available distributions and its syntax are:

• "normal(mu,sd)" with mu and sd being floating numbers: Normal distribution with mean mu and standard deviation sd.

• "normalCapped(mu, sd, min, max)" By default, no negative values are accepted but may be enabled by setting a negative lower limit.

• "lognormal(mu,sd)" with mu and sd being floating numbers: Normal distribution with mean mu and standard deviation sd.

• "uniform(a,b)" with limits a and b being floating numbers: Uniform distribution between a and b.

• "gamma(alpha,beta)" with parameters alpha and beta: Gamma distribution.

The command is shown below.

```

python tools/createVehTypeDistribution.py config.txt

```

The output file looks as following:

```xml

<xml version="1.0"?>

<additional xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/additional\_file.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<vTypeDistribution id="passenger">

<vType id="passenger0" vClass="passenger" tau="1.200" speedFactor="normc(1,0.11,0.2,2)" sigma="0.055" minGap="1.200" maxSpeed="80.000" length="4.949" guiShape="passenger" decel="5.000" carFollowModel="Krauss" accel="3.000"/>

<vType id="passenger1" vClass="passenger" tau="1.200" speedFactor="normc(1,0.11,0.2,2)" sigma="0.135" minGap="1.200" maxSpeed="80.000" length="4.540" guiShape="passenger" decel="5.000" carFollowModel="Krauss" accel="3.000"/>

<vType id="passenger2" vClass="passenger" tau="1.200" speedFactor="normc(1,0.11,0.2,2)" sigma="0.044" minGap="1.200" maxSpeed="80.000" length="4.872" guiShape="passenger" decel="5.000" carFollowModel="Krauss" accel="3.000"/>

<vType id="passenger3" vClass="passenger" tau="1.200" speedFactor="normc(1,0.11,0.2,2)" sigma="0.040" minGap="1.200" maxSpeed="80.000" length="4.902" guiShape="passenger" decel="5.000" carFollowModel="Krauss" accel="3.000"/>

```

\*\*Additional options:\*\*

--output-file configures the name of the output file to be written

--name Name of the created distribution

--size Number of s to be sampled for filling the distribution

--seed Set the seed for the random number generator

### Step 11.2. Mesoscopic model

[MESO]( https://sumo.dlr.de/docs/Simulation/Meso.html) refers to a mesoscopic simulation model which uses the same input data as the main sumo model. It computes vehicle movements with queues and runs up to 100 times faster than the microscopic model of sumo. Additionally, due to using a coarser model for intersections and lane-changing, it is more tolerant of network modeling errors than sumo.

The Mesoscopic model calibrates with a Microscopic model with a measure [Fréchet distance](https://en.wikipedia.org/wiki/Fr%C3%A9chet\_distance), [T-test](https://en.wikipedia.org/wiki/Student%27s\_t-test) , and RMSE [[2]](https://ops.fhwa.dot.gov/publications/fhwahop18036/chapter3.htm#simulation-run-control-data), [[3]](https://static.tti.tamu.edu/tti.tamu.edu/documents/4198-2.pdf).

Only a few [vType](https://sumo.dlr.de/docs/Definition\_of\_Vehicles%2C\_Vehicle\_Types%2C\_and\_Routes.html) parameters affect the mesoscopic simulation. They are listed below:

• [vClass](https://sumo.dlr.de/docs/Definition\_of\_Vehicles%2C\_Vehicle\_Types%2C\_and\_Routes.html#vehicle\_types)

• [length](https://sumo.dlr.de/docs/Definition\_of\_Vehicles%2C\_Vehicle\_Types%2C\_and\_Routes.html#vehicle\_length)

• [minGap](https://sumo.dlr.de/docs/Definition\_of\_Vehicles%2C\_Vehicle\_Types%2C\_and\_Routes.html#vehicle\_length)

• [maxSpeed](https://sumo.dlr.de/docs/Vehicle\_Type\_Parameter\_Defaults.html)

• [speedFactor, speedDev](https://sumo.dlr.de/docs/Simulation/Meso.html#further\_congestion\_effects )

• [Impatience]( https://sumo.dlr.de/docs/Simulation/Meso.html#impatience )

• [accel,decel](https://sumo.dlr.de/docs/Vehicle\_Type\_Parameter\_Defaults.html) (only for computing junction passing time when the microscopic junction model is active).

Use python script [createVehTypeDistribution.py]( https://sumo.dlr.de/docs/Tools/Misc.html ) to generate a vehicle type distribution by sampling from configurable value distributions for the desired vType-parameters.

##Step 12. Configure and run Micro-SUMO simulation

Set up the configuration file with set of parameters as [vTypeDistributions\_micro.add.xml]() and output parameters: [Chattanooga\_SUMO\_calibration\_final.sumocfg]().

This configuration will generate three output files:

\* Edge-based output.

\* Detectors output file [out.xml](), contains 137 detectores wtith flow and speed data to compare with real-world speed data.

\* Trajectory output which contains information about type, current speed and acceleration of each vehicle.

\*Note:\* [Chattanooga\_Daily\_Trips\_calibtation.rou.xml](), which is too large to push here, is put in Teams.

##Step 13. Configure and run Meso-SUMO simulation

Set up the configuration file with set of parameters as [vTypeDistributions\_meso.add.xml]() and output parameters: [Chattanooga\_SUMO\_calibration\_final.sumocfg]().

This configuration will generate three output files:

\* Edge-based output.

\* Detectors output file [out.xml](), contains 137 detectores wtith flow and speed data to compare with real-world speed data.

\* Trajectory output which contains information about type, current speed and acceleration of each vehicle.

\*Note:\* [Chattanooga\_Daily\_Trips\_calibtation.rou.xml](), which is too large to push here, is put in Teams.

##Step 14. Process the output files

For calibration, Micro-SUMO simulation results should be compared with the real-world with a T-test and [Fréchet distance](http://www.kr.tuwien.ac.at/staff/eiter/et-archive/cdtr9464.pdf) to find which speed result is close to the real-world data.

• For the T-test, first, determine a significance level appropriate for the study. This will be the value for comparing the final result. Generally, significance values are at α = 0.05 or α = 0.01, depending on the preference and how accurate the results are.

• Compute the discrete Fréchet distance between two curves. Use python scripts [similaritymeasures](https://pypi.org/project/similaritymeasures/) and [frechetdist](https://pypi.org/project/frechetdist/) to calculate the minimum distance between two curves.

The process should be repeated for calibration of the Meso-SUMO model between the result of Meso and Micro.

\*\*Click the above blue highlight texts for more information.\*\*