# Using SUMO to Validate Mixed Autonomy Road Capacity Models

David Rower
*Department of Physics, University of California Santa Barbara, Santa Barbara CA 93106*

## I. MOTIVATION AND SCOPE

This report was created with two goals: to detail some simple simulation studies validating two mixed autonomy road capacity models presented in Lazar et al. [1] and to provide some basic knowledge of the microscopic traffic simulator SUMO used to validate the models. We present a brief overview of the capacity models and SUMO configuration, and a terse discussion of some simulation artifacts. We also offer a guide to replicate the specific build of SUMO used in these studies in the appendix.

## II. ROAD CAPACITY MODELS

### A. Capacity Model 1

In this model, an autonomous vehicle is indiscriminate in reducing its headway when following other vehicles. Let $m$ be the capacity of the road when fully utilized by regular vehicles, and $M$ be the capacity of the road when fully utilized by autonomous vehicles. Let $\alpha$ be the average proportion of smart vehicles on the road. The capacity of the road under autonomy level $\alpha$ is approximated by

$$C(\alpha) = \frac{1}{\alpha M^{-1} + (1-\alpha)m^{-1}}. \tag{1}$$

Note that, for a single lane road of length $d$, where regular vehicles assume headway $h_r$ and autonomous vehicles assume headway $h_s$, we have $M = d/h_s$ and $m = d/h_r$. Let the length of every vehicle be $l$. We can then express the capacity as

$$C(\alpha) = \frac{d}{\alpha h_s + (1-\alpha)h_r + l}. \tag{2}$$

The physical significance of $h_r$ and $h_s$ can be appreciated below in Fig. 1.

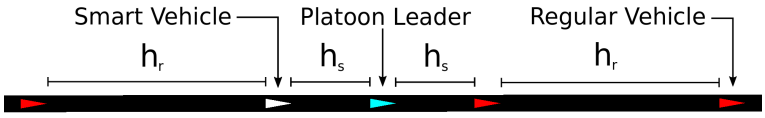


FIG. 1: A representation of car interactions in SUMO according to capacity model 1. In this model, an autonomous vehicle will always follow with a distance $h_s$ to the car in front of it. A regular vehicle will follow with headway $h_r$.

### B. Capacity Model 2

In this model, an autonomous vehicle adjusts its headway according to the technology of the car it is following; it only reduces its headway when following another autonomous vehicle. Using

the same notation as above, we have

$$C(\alpha) = \frac{1}{\alpha^2 M^{-1} + (1-\alpha^2)m^{-1}} = \frac{d}{\alpha^2 h_r + (1-\alpha^2)h_s + l}. \tag{3}$$

The physical significance of $h_r$ and $h_s$ can be appreciated in Fig. 2 below.

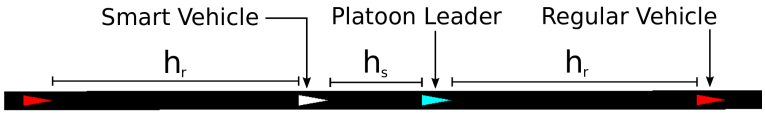


FIG. 2: A representation of car interactions in SUMO according to capacity model 2. In this model, an autonomous vehicle will only follow with a distance $h_s$ to the car in front of it if that car is also autonomous. A regular vehicle will always be followed with distance $h_r$.

## III. VALIDATION OF MODELS IN SUMO

### A. SUMO Configuration

The most basic road network was used to validate the capacity models - a single-lane road in a straight line. Below, we outline the basic configuration files needed to define this scenario.

1. `/single_road/network/single.net.xml`
   - This file defines the road graph, including the locations of vertices in a plane, the edges between vertices, and the speed limit along those edges.
2. `/single_road/network/single.rou.xml`
   - Types of vehicles are defined here according to a car following model, color, acceleration parameters, impatience, length of vehicle, maximum speed, minimum gap from vehicle immediately in front, and headway.
   - Instances of these vehicles on the road are also defined here. This can be done in multiple ways, but for this study it was useful to define a "flow," where a proportion of different vehicle types is specified and the traffic is generated from this distribution.
3. `/single_road/network/single.det.xml`
   - This file defines sensors on the road, including the locations of the sensors on specific edges of the road graph, the frequency of detection, and the output files. Note that the output files weren't necessary, as an interface from Python was available to directly talk to these sensors.

An example of a sensor in the graphical rendering of SUMO is presented in Fig. 3.


FIG. 3: A sensor as displayed in the GUI of SUMO. The yellow box is the sensor, which can be listened to directly through a Python script. The red triangle represents a car.

### B. Overview of TraCI

In order to gather data quickly and effectively from SUMO, TraCI, a "Traffic Control Interface," was released. TraCI allows you to directly observe and manipulate instances of SUMO via several supported programming languages, including Python. This was used to automate the running of several instances of SUMO, and to collect data from each of those instances.

### C. Our Scenario

A single-lane road in a straight line was defined, and two sensors were defined along the road. One sensor was placed roughly 1/5 of the way down the road, the other was placed near the end of the road (but not at the end, as this led to a bug). The first sensor was placed such that the traffic flow from the source node would equilibrate before reaching the sensor. The number of cars which passed each sensor could be counted, and the number of cars on the patch of road between the two sensors could be calculated from the difference between the sensor counts.

### D. Methodology

This number of cars on the patch between the sensors was recorded several times over the duration of a simulation, and the average and standard deviation were computed. The average number of cars on the road was plotted against the road capacity for a sampling of autonomy levels for two scenarios: parameterizations matching the descriptions of capacity models 1 and 2. These results are plotted in Fig. 4 and Fig. 5.

### E. Discussion

There are two common features in these studies. The standard deviation of the road capacity is a maximum when the traffic is very mixed. It takes on low values when the traffic tends to be very regular or very autonomous. The second common feature is an artifact of the finite length of the road and only affects the simulations for autonomy level $\alpha = 1$.

As vehicles are added to the road, they need to accelerate to catch up to the (infinite) platoon in front of them. Over time, the gap between a newly injected vehicle and the vehicle injected before it grew too large to be overcome by the acceleration of the vehicles. This problem wasn't present in simulations with lower values of $\alpha$ since regular vehicles would be injected often enough to keep this gap from growing too large.

### F. Conclusion

In summary, a brief overview of two capacity models for mixed autonomy traffic was presented. A SUMO scenario to validate these models was described, and simulation results were presented. There was strong agreement between both models and the simulations using their corresponding parameterizations for autonomous vehicle behavior for all autonomy levels $\alpha \neq 1$. For the case $\alpha = 1$, an artifact of the finite length of the road created a disagreement between the capacity models' predictions and the simulation results for both models.

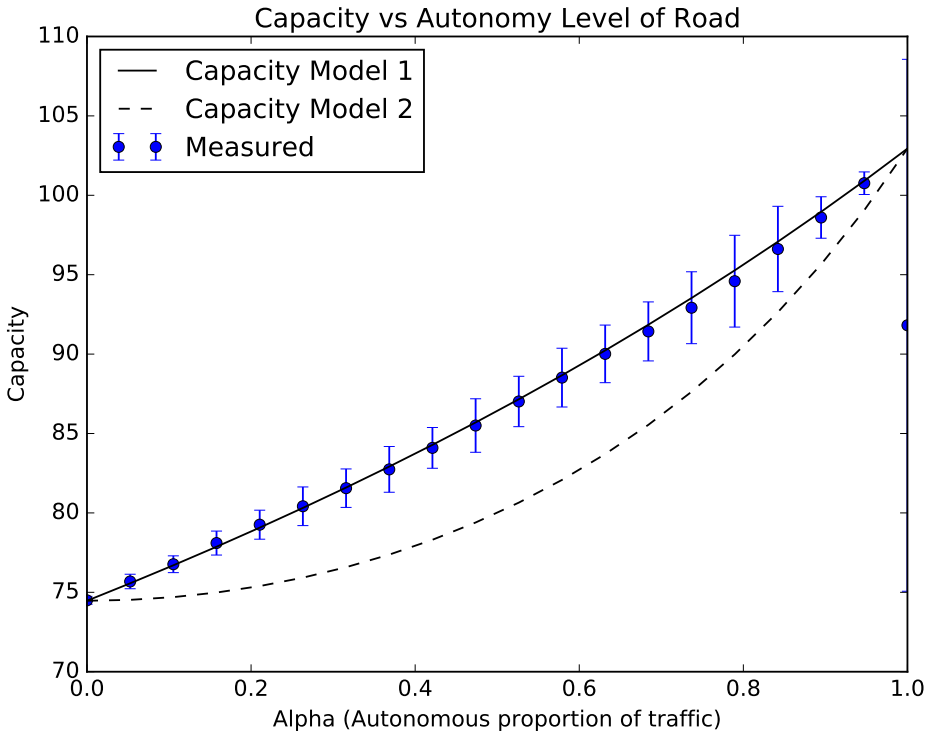


FIG. 4: Road capacity as a function of autonomy level with parameterization of capacity model 1. In this model, autonomous vehicles are indiscriminate in reducing their headway. There is a very strong agreement between the model and the measurements for all values of $\alpha \neq 1$.

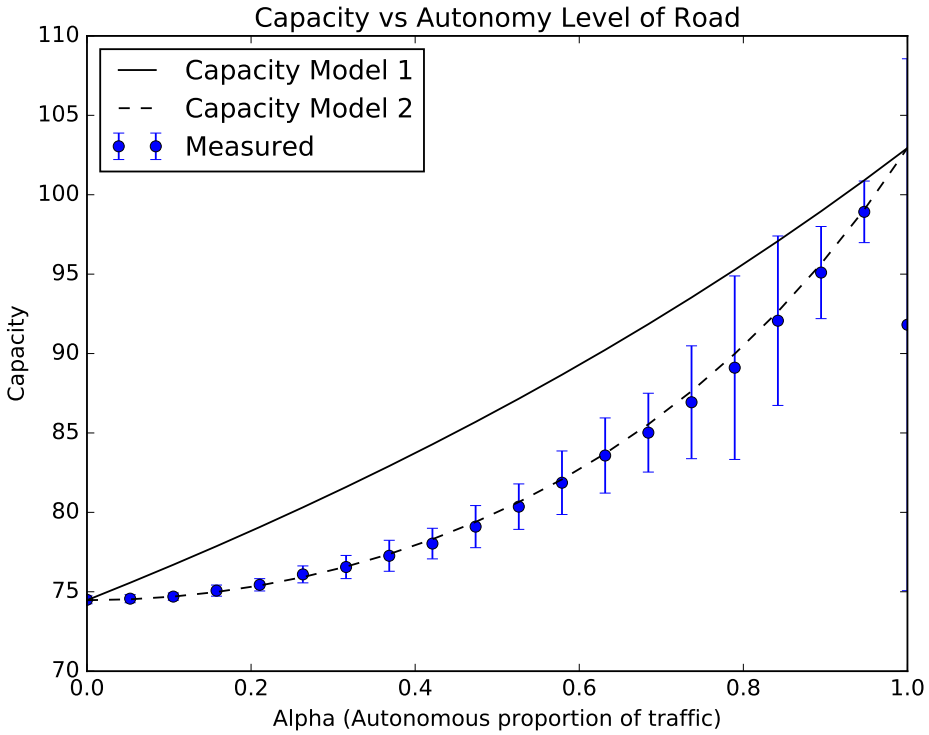


FIG. 5: Road capacity as a function of autonomy level with parameterization of capacity model 2. In this model, autonomous vehicles only reduce their headway if the leading car is also autonomous. There is a very strong agreement between the model and the measurements for all values of $\alpha \neq 1$.

## References

[1] D. Lazar, S. Coogan, R. Pedarsani, "Capacity Modeling and Routing for Traffic Networks with Mixed Autonomy," *IEEE Conference on Decision and Control (CDC)*, 2017.
[2] D. Krajzewicz, J. Erdmann, M. Behrisch, L. Bieker. "Recent Development and Applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, 5 (34):128-138, 2012.
[3] A. Askari, D. Farias, A. Kurzhanskiy, P. Varaiya, "Effect of Adaptive and Cooperative Adaptive Cruise Control on Throughput of Signalized Arterials," *arXiv:1703.01657v1*, 2017.

## Appendix A: Building Sumo with /ucbtrans/sumo-project

### 1. Warnings

I have only built SUMO with the additions in Ubuntu 16.04, so be warned this may not work on all other systems. Also be warned, it may be possible to get around using some of the mentioned dependencies by disabling certain features of SUMO, or it may be possible to use different packages to suffice for the dependencies, but this worked for me.

### 2. Setting up the directories

First, you need to download the repository source files and SUMO 0.30.0. After extracting the files from the Github repository, you can find a directory `sumo-project-master/sumo-0.30.0`, in which you should copy the source files for SUMO 0.30.0. You can simply drag the SUMO 0.30.0 source folder into the sumo-project-master directory, and a reasonable GUI will ask you if you'd like to merge the folders. Make sure **not** to replace the sumo-project-master files, as these add the IIDM functionality we desire.

### 3. Dependencies

Now we are almost ready to compile the project. There are only two dependencies (that I did not already have on a relatively clean system) required to do this. The first is Xerces-C++ (some kind of XML parser). You can install it with the package manager.

```
sudo apt-get install libxerces-c-dev
```

The second dependency is FOX (some kind of GUI toolkit), which can also be installed via the package manager.

```
sudo apt-get install libfox-1.6-dev
```

### 4. Compiling the project

To compile the project, you first need to cd into the `sumo-project-master/sumo-0.30.0` directory.

```
cd ~/<specific to you>/sumo-project-master/sumo-0.30.0
```

Then, you need to run the `./configure` command.

```
./configure
```

Finally, we are ready to make the project. There are a few optional arguments to the make command, although I am not quite sure what we need yet, so I have just used the default settings.

```
make
```

If this works successfully, pat yourself on the back. Then, before you get too excited, remember to add the environment variable to your `~/.bashrc` file, so that you can communicate to SUMO with Python. To do this, simply open your bash profile.

```
gedit ~/.bashrc
```

and paste this line into the text file

```
export SUMO_HOME=$HOME/<specific to you>/sumo-project-master/sumo-0.30.0
```

Also recommended: paste this line, which will allow you to run SUMO from the command line

```
export PATH=$PATH:$HOME/<specific to you>/sumo-project-master/sumo-0.30.0/bin
```

## 5. Running Examples

There are Python scripts that run each example, each with a name that starts with "runner." These files can be executed directly if you change the file access permissions to add executability.

```
chmod +x runner.py
```

Then you can simply run the example.

```
./runner.py
```

Some of these examples will work right off the bat - like `/example/redLight_simulations`. Some will not.

### *a. Platoon Bug Fix*

One of the cases I have been able to debug is those examples that use platooning functions. If you run the script without modifying the runner file, you may run into an error complaining that the TraCI module doesn't exist.

The bug fix is simple - in the `runner.py` file, near the beginning, you will find a line.

```
from platoon_functions import *
```

In the platoon_functions file, TraCI is imported, although Python does not yet know where the TraCI module can be found. In the original runner file, after the line shown above, you will see a

big try/except statement resembling

```
# import python modules from $SUMO_HOME/tools directory
try:
    ...
except ImportError:
    ...
```

This block tells Python where to find your SUMO files. Move the line

```
from platoon_functions import *
```

to be below that try/except statement. This should resolve the error and allow TraCI to be imported in the platoon_functions module.