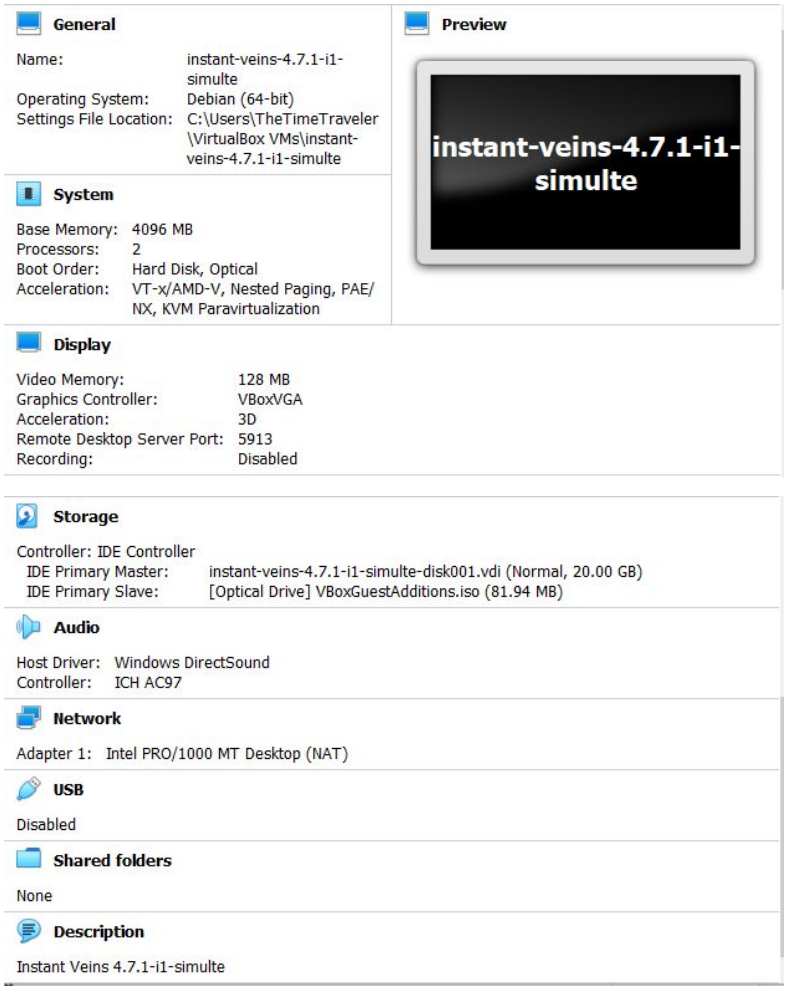
Getting Started With SUMO using Instant Veins

1. Download and install **Instant Veins 4.7.1 i1 with SimuLTE** by using a virtual machine software such as **Oracle VM Virtualbox 6.08.**
   https://veins.car2x.org/download/
   https://www.virtualbox.org/wiki/Downloads

2. Set up the settings of the Virtual Machine with following settings.



3. This VM comes with the sumo-0.32.0 version installed. In order to install the rogue car/DSRC patch for this current project, you could install the patch at:
   https://github.com/floreezyy/cmps116-IoT/tree/master/patch
   By placing the file.patch file into your sumo source directory and running the command:
   ```
   $ patch -t -p0 <file.patch
   ```
   Unfortunately, patching does not affect the binary files of a source folder, only the source code, which means that after installing the patch, you will need to:

```
$ cd sumo-0.32.0
$ make
```

Which will take 30-35 minutes. Or you could navigate to: https://github.com/floreezyy/cmps116-IoT/tree/master/current_build/ Delete the source folder that comes with SUMO (sumo-0.32.0), then copy and paste our current build into your source directory, which is much quicker.

4. Create a new directory for your SUMO simulation files.
5. To run a simulation, you need a .net.xml file, a .rou.xml file, and a .sumocfg file within this directory. A user can have these files automatically generated, or write them individually.
6.

Getting Started With SUMO from Scratch Using Linux

Automatic Generation

Using OSM to generate a Roadway Network

Export/download a target map (.osm file) from https://www.openstreetmap.org/. Make sure the map you are exporting is not too large, or the site will crash; pick a block from a random metropolitan city. Refer to this link for additional help: https://sumo.dlr.de/userdoc/Tutorials/Import_from_OpenStreetMap.html

Run the following command after you download a map.osm file:

```
$ netconvert --osm-files my_map.osm -o my_map.net.xml
--geometry.remove --ramps.guess --junctions.join --
tls.guess-signals --tls.discard-simple --tls.join
--remove-edges.isolated --tls.half-offset
```

Each option, in order, removes complexities from the network topology without changing it. The commands add ramp lanes, merge junctions automatically, transcribe traffic light information, discard traffic light information, join traffic light info, discard any edges without a connection, shift traffic light timing by half phase, and remove bicycle lanes. A .net.xml file is generated according to what the user named it, in this case, my_map.net.xml.

Check to see that your **.net.xml file** was generated properly using the following command in your terminal, which will bring up the map in sumo-gui with no vehicles running:

```
$ sumo-gui -n my_net.net.xml
```

As an example of all of the things you can do with the randomTrips.py python script, the command below generates 40 random trips on my_net.net.xml, 10% (4) of which will be rogue, 20% (8) of which will be passenger, and 70% (28) of which will be emergency vehicles, each with an attribute that their acceleration ability is 1 m/s^2 and their color is red. Only the rogue and passenger vehicles will have DSRC devices attached to them (and therefore transmit packets to the RSU's on the network), while the emergency vehicles will not have DSRC devices:

```
$ ~/src/sumo/tools/randomTrips.py -n my_net.net.xml -e 40
--vehicle-class rogue --vehicle-percentage 10 --device dsrc
--vehicle-class passenger --vehicle-percentage 20 --device dsrc
--vehicle-class emergency --vehicle-percentage 70 --device none
--trip-attributes="accel=\"1.0\" color=\"red\""
```

Vehicle types and parameters are explained in detail at:
https://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes

Lastly, the user must create a config file such as my_config.sumocfg below:

```
<configuration>
    <input>
      <net-file value="my_net.net.xml"/>
      <additional-files value="trips.trips.xml"/>
    </input>
    <time>
    <begin value="0"/>
    <end value="2000"/>
    </time>
</configuration>
```

This puts all of a simulation's files in one referrable place. Run the following command to run a simulation on your map with randomly generated trips:

```
$ sumo-gui my_config.sumocfg
```

When you create trips using randomTrips.py, it chooses a unique route at random starting points across the map, keeping traffic congestion in mind.

Refer to this link for more information on randomTrips.py (only covering the features not implemented by us):
http://sumo.sourceforge.net/userdoc/Tools/Trip.html

Refer to this link for more information on controlling routes dynamically (ex. Ignore route errors):
https://sumo.dlr.de/wiki/Demand/Automatic_Routing

Manual Generation

.net.xml files can be created manually as well, defining nodes and junctions in a **.nod.xml** file, edges or connected junctions in an **.edg.xml** file, and, a **.type.xml** file to list out lane types concisely.

Refer to the following link for more information:
https://sumo.dlr.de/wiki/Networks/PlainXML

Example of a node file is located in:
https://github.com/floreezyy/cmps116-IoT/blob/master/dev/rkhamamo/light_run/my_nod.nod.xml

Nodes are junctions and you have a unique id, x/y coordinates, priority type attributes for the edges.

```
<nodes>
<node id="n1" x = "-500" y="0" type="priority"/>
<node id="n2" x = "-250" y="0" type="traffic_light"/>
<node id="n3" x = "-250" y="200"/>
<node id="n4" x = "-250" y="-200"/>
<node id="n5" x = "0" y="200" type="traffic_light"/>
<node id="n6" x = "0" y="-200" type="traffic_light"/>
<node id="n7" x = "0" y="400"/>
<node id="n8" x = "0" y="-400"/>
<node id="n9" x = "250" y="400"/>
<node id="n10" x = "250" y="-400"/>
<node id="n11" x = "0" y="0" type="traffic_light"/>
```

```
<node id="n12" x = "250" y="0"/>
</nodes>
```

Example of a edge file is located in:
https://github.com/floreezyy/cmps116-IoT/blob/master/dev/rkhamamo/light_run/my_edg.edg.xml

You have a unique node ID that has a (from - to) with a custom type describing # of lanes and speed limit.

```
<edges>

 <edge from="n1" to="n2" id="1to2" type="2L45"/>
 <edge from="n2" to="n1" id="2to1" type="2L45"/>

 <edge from="n2" to="n3" id="2to3" type="2L5"/>
 <edge from="n3" to="n2" id="3to2" type="2L5"/>

 <edge from="n2" to="n4" id="2to4" type="2L5"/>
 <edge from="n4" to="n2" id="4to2" type="2L5"/>

 <edge from="n3" to="n5" id="3to5" type="2L30"/>
 <edge from="n5" to="n3" id="5to3" type="2L30"/>

 <edge from="n5" to="n7" id="5to7" type="2L30"/>
 <edge from="n7" to="n5" id="7to5" type="2L30"/>

 <edge from="n5" to="n11" id="5to11" type="2L30"/>
 <edge from="n11" to="n5" id="11to5" type="2L30"/>

 <edge from="n4" to="n6" id="4to6" type="2L30"/>
 <edge from="n6" to="n4" id="6to4" type="2L30"/>

 <edge from="n6" to="n8" id="6to8" type="2L30"/>
 <edge from="n8" to="n6" id="8to6" type="2L30"/>

 <edge from="n6" to="n11" id="6to11" type="2L30"/>
 <edge from="n11" to="n6" id="11to6" type="2L30"/>
```

```
<edge from="n7" to="n9" id="7to9" type="2L30"/>
<edge from="n9" to="n7" id="9to7" type="2L30"/>

<edge from="n11" to="n12" id="11to12" type="2L30"/>
<edge from="n12" to="n11" id="12to11" type="2L30"/>

<edge from="n8" to="n10" id="8to10" type="2L30"/>
<edge from="n10" to="n8" id="10to8" type="2L30"/>

</edges>
```

An example of a .type.xml file is located at https://github.com/floreezyy/cmps116-IoT/blob/master/dev/rkhamamo/light_run/my_type.type.xml

In order to have custom type in the edges file, you need to also add a type file:

```
<types>
<type id="2L45" priority="3" numLanes="2" speed="45"/>
<type id="2L5" priority="3" numLanes="2" speed="5"/>
<type id="2L30" priority="2" numLanes="2" speed="30"/>
</types>
```

After creating a (**.nod.xml**), (**.edg.xml**) and (**.type.xml**) files you can create **.net.xml** file using

```
$ netconvert --node-files=my_nod.nod.xml
--edge-files=my_edg.edg.xml
--type-files=my_type.type.xml --output-file=my_net.net.xml
```

Refer to this link https://sumo.dlr.de/wiki/NETCONVERT
For more options using netconvert.

If you want to further edit your network file, such as fixing problematic edges or adding/ modifying traffic lights in a uncomplicated way, you can use netedit

Refer to this link https://sumo.dlr.de/wiki/NETEDIT

Create a polygon file to import additional polygons to make the simulation more visually appealing. This file simply just adds color to different aspects of the map that is already in the .osm file data. Notice how its on the -4 layer so its at the very bottom of the GUI.

```
$ polyconvert --net-file my_net.net.xml --osm-files my_osm.osm
typemap.xml -o my_poly.poly.xml
```

Use typemap.xml file from:
/home/veins/src/sumo-0.32/data/typemap/osmPolyconvert.typ.xml

Typemap.xml

```
<polygonTypes>
    <polygonType id="waterway"              name="water"
color=".71,.82,.82" layer="-4"/>
    <polygonType id="natural"               name="natural"
color=".55,.77,.42" layer="-4"/>
    <polygonType id="natural.water"         name="water"
color=".71,.82,.82" layer="-4"/>
    <polygonType id="natural.wetland"       name="water"
color=".71,.82,.82" layer="-4"/>
    <polygonType id="natural.wood"          name="forest"
color=".55,.77,.42" layer="-4"/>
    <polygonType id="natural.land"          name="land"
color=".98,.87,.46" layer="-4"/>

    <polygonType id="landuse"               name="landuse"
color=".76,.76,.51" layer="-3"/>
    <polygonType id="landuse.forest"        name="forest"
color=".55,.77,.42" layer="-3"/>
    <polygonType id="landuse.park"          name="park"
color=".81,.96,.79" layer="-3"/>
```

```
    <polygonType id="landuse.residential"     name="residential"
color=".92,.92,.89" layer="-3"/>
    <polygonType id="landuse.commercial"      name="commercial"
color=".82,.82,.80" layer="-3"/>
    <polygonType id="landuse.industrial"      name="industrial"
color=".82,.82,.80" layer="-3"/>
    <polygonType id="landuse.military"        name="military"
color=".60,.60,.36" layer="-3"/>
    <polygonType id="landuse.farm"            name="farm"
color=".95,.95,.80" layer="-3"/>
    <polygonType id="landuse.greenfield"      name="farm"
color=".95,.95,.80" layer="-3"/>
    <polygonType id="landuse.village_green"   name="farm"
color=".95,.95,.80" layer="-3"/>

    <polygonType id="tourism"                 name="tourism"
color=".81,.96,.79" layer="-2"/>
    <polygonType id="military"                name="military"
color=".60,.60,.36" layer="-2"/>
    <polygonType id="sport"                   name="sport"
color=".31,.90,.49" layer="-2"/>
    <polygonType id="leisure"                 name="leisure"
color=".81,.96,.79" layer="-2"/>
    <polygonType id="leisure.park"            name="tourism"
color=".81,.96,.79" layer="-2"/>
    <polygonType id="aeroway"                 name="aeroway"
color=".50,.50,.50" layer="-2"/>
    <polygonType id="aerialway"               name="aerialway"
color=".20,.20,.20" layer="-2"/>

    <polygonType id="shop"                    name="shop"
color=".93,.78,1.0" layer="-1"/>
    <polygonType id="historic"                name="historic"
color=".50,1.0,.50" layer="-1"/>
    <polygonType id="man_made"                name="building"
color="1.0,.90,.90" layer="-1"/>
    <polygonType id="building"                name="building"
color="1.0,.90,.90" layer="-1"/>
```

```
    <polygonType id="amenity"                 name="amenity"
color=".93,.78,.78" layer="-1"/>
    <polygonType id="amenity.parking"         name="parking"
color=".72,.72,.70" layer="-1"/>
    <polygonType id="power"                   name="power"
color=".10,.10,.30" layer="-1" discard="true"/>
    <polygonType id="highway"                 name="highway"
color=".10,.10,.10" layer="-1" discard="true"/>

    <polygonType id="boundary" name="boundary"
color="1.0,.33,.33" layer="0" fill="false" discard="true"/>
    <polygonType id="admin_level" name="admin_level"
color="1.0,.33,.33" layer="0" fill="false" discard="true"/>
</polygonTypes>
```

7. To check to see your **.net.xml file** road network just run the command

```
sumo-gui -n hello.net.xml
```

8. Next, you need to add vehicles to your road network and to do this you need to add a **(.rou.xml)** file. Below is an example of a simple route file.

```
<routes>
    <vType accel="1.0" decel="5.0" id="Car" length="2.0"
maxSpeed="100.0" sigma="0.0" />
    <route id="route1" edges="1to2 2to3 3to5 5to7 7to9"/>
    <vehicle depart="10" id="veh2" route="route1" type="Car" />
</vehicle>

</routes>
```

These files create a vehicle, a route for the vehicle to follow, and a start time. Refer to the link below for an explanation of the different parameters you can add to a vehicle, as well as the variety of declarable vehicle classes:

https://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes

A new vehicle class, the rogue car class, is declarable with the new release, characterized by doubling the speed limit, not respecting minimum follow distance,

merging instantly when the rogue vehicle must decelerate for another vehicle, and running red lights if it does not pose a collision risk. The vehicle will not intentionally collide with other cars, and will follow traffic laws if it is surrounded by regular traffic. It will not run red lights when behind another car without a means of merging around it. Within the gui, the vehicle appears as a regular passenger car unless other characteristics are specified. To instantiate a rogue vehicle, insert the `vClass ="rogue"` parameter into the route file where vehicles are declared.

Here is an example of a route file with DSRC devices and Rogue car type. This will be explained later in the document

https://github.com/floreezyy/cmps116-IoT/blob/master/dev/rkhamamo/light_run/my_rou.rou.xml

```
<routes>
<vType vClass ="passenger" accel="1.0" decel="5.0" id="Car"
length = "2.0" maxSpeed="100.0" sigma="0.0" />
<vType vClass ="rogue" accel="1.0" decel="5.0" id="Rogue" length
= "2.0" maxSpeed="100.0" sigma="0.0" />

<route id="route1" edges="1to2 2to3 3to5 5to7 7to9"/>

<vehicle depart="10" id="veh2" route="route1" type="Car">
    <param key="has.dsrc.device" value="true"/>
</vehicle>
<vehicle depart="15" id="veh3" route="route1" type="Car">
    <param key="has.dsrc.device" value="true"/>
</vehicle>


<vehicle depart="20" id="veh1a" route="route1" type="Rogue"
color = "255,0,0,0">
    <param key="has.dsrc.device" value="true"/>
</vehicle>
<vehicle depart="20" id="veh1b" route="route1" type="Rogue"
color = "255,0,0,0">
    <param key="has.dsrc.device" value="true"/>
</vehicle>
```

```
<vehicle depart="20" id="veh1" route="route1" type="Rogue" color
= "255,0,0,255">
    <param key="has.dsrc.device" value="true"/>
</vehicle>
<vehicle depart="20" id="veh1c" route="route1" type="Rogue"
color = "255,0,0,0">
    <param key="has.dsrc.device" value="true"/>
</vehicle>
<vehicle depart="20" id="veh1d" route="route1" type="Rogue"
color = "255,0,0,0">
    <param key="has.dsrc.device" value="true"/>
</vehicle>

<vehicle depart="25" id="veh4" route="route1" type="Car" >
    <param key="has.dsrc.device" value="true"/>
</vehicle>
<vehicle depart="30" id="veh5" route="route1" type="Car" >
    <param key="has.dsrc.device" value="true"/>
</vehicle>
<!--<route id="route2" edges="1to2 2to4 4to6 6to8 8to10"/>-->
<!--<vehicle depart="20" id="veh2" route="route2"
type="Ambulance" />-->

<!--<route id="route3" edges="1to2 2to3 3to5 5to11 11to12"/>-->
<!--<vehicle depart="30" id="veh3" route="route3" type="Car"
/>-->

<!--<route id="route4" edges="1to2 2to4 4to6 6to11 11to12"/>-->
<!--<vehicle depart="40" id="veh4" route="route4" type="Car"
/>-->

</routes>
```

9. To add Roadside Units and Rerouters to the simulation in order to make sure the car does not go out of the map, do the following

Create a **.type.xml** file

The fields include a polygon type, color, rectangle shape points, rerouter at a specific edge, and RSU detector at a specific lane.

An example of a .type.xml file is located at https://github.com/floreezyy/cmps116-IoT/blob/master/dev/rkhamamo/light_run/my_add.add.xml

```
<additionals>

    <poly id="1" type = "building" color="255,230,230" fill="1"
layer="-1.00" shape="1952.28,1419.89 1966.35,1420.19
1966.85,1396.76 1952.78,1396.46 1952.28,1419.89"/>

    <rerouter id="rerouter_0" edges="3to5" probability="1">
        <interval begin="0" end="10000">
             <destProbReroute id="5to11"/>
          </interval>
    </rerouter>

    <rerouter id="rerouter_2" edges="5to11" probability="1">
        <interval begin="0" end="10000">
             <destProbReroute id="11to6"/>
          </interval>
    </rerouter>

    <rerouter id="rerouter_2" edges="11to6" probability="1">
        <interval begin="0" end="10000">
             <destProbReroute id="6to4"/>
          </interval>
    </rerouter>

    <rerouter id="rerouter_2" edges="6to4" probability="1">
        <interval begin="0" end="10000">
             <destProbReroute id="4to2"/>
          </interval>
    </rerouter>

    <rerouter id="rerouter_2" edges="4to2" probability="1">
        <interval begin="0" end="10000">
             <destProbReroute id="2to3"/>
```

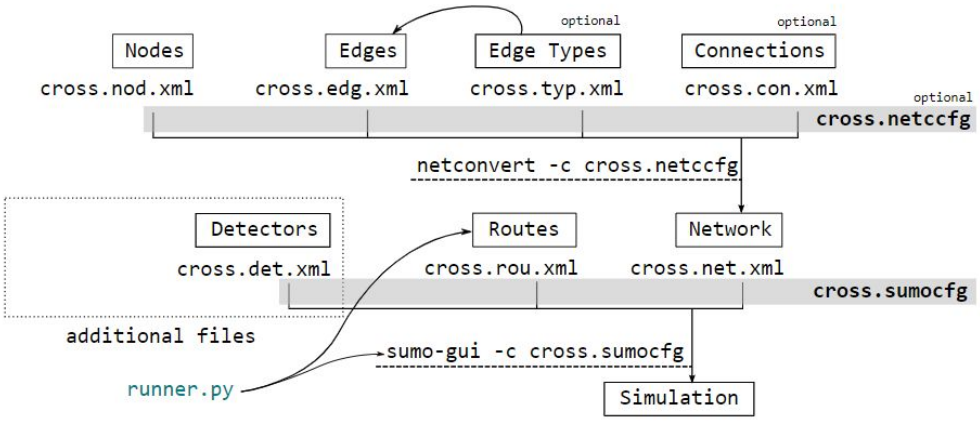
```
        </interval>
    </rerouter>

    <rerouter id="rerouter_2" edges="2to3" probability="1">
        <interval begin="0" end="10000">
            <destProbReroute id="3to5"/>
        </interval>
    </rerouter>
    <e1Detector id="rsu_1" lane="3to5_0" pos="100" freq="500"
file="my_out.out.xml" />
    <e1Detector id="rsu_2" lane="11to6_0" pos="100" freq="500"
file="my_out.out.xml" />
    <e1Detector id="rsu_3" lane="4to2_0" pos="100" freq="500"
file="my_out.out.xml" />

</additionals>
```

10. Finally, you need to create a sumo configuration file.

An example of a sumo cfg file is located at https://github.com/floreezyy/cmps116-IoT/blob/master/dev/rkhamamo/light_run/my_config.sumocfg

```
<configuration>
     <input>
          <net-file value="my_net.net.xml"/>
          <route-files value="my_rou.rou.xml"/>
          <additional-files value="my_add.add.xml"/>
          <step-length value="0.01"/>
     </input>
     <time>
     <begin value="0"/>
     <end value="2000"/>
     </time>
</configuration>
```

11. To dive deep and to get an overview of all these steps and also get an introduction to Traci, you can refer to this helpful link.



https://intelaligent.github.io/tctb/post-learning-traci-tls.html

TraCI

Getting Started

TraCI (Traffic Control Interface) allows a user to dynamically affect a running SUMO simulation and retrieve data values of simulated objects. TraCI uses a TCP based client/server architecture to provide access to SUMO, with a python script acting as the client, and the SUMO simulation acting as the server. The TraCI python script imports functions from the tools folder (./sumo/tools/traci) of the user's SUMO base qdirectory, set as SUMO_HOME. For example, in the instant Veins IDE, SUMO_HOME must be defined in the operating system's Z-shell resource (~/.zshrc) via the line:

```
export SUMO_HOME="/home/veins/src/sumo-0.32.0"
```

Write this line anywhere in the .zshrc file, then subsequently either reboot the IDE or use the following command to implement changes in the current session:

```
$ source ~/.zshrc
```

The script then begins SUMO as a subprocess, then connects to sumo and runs depending on the command line configuration.

Writing the Script

For a detailed description of how to write a traci pythons script, refer to:
http://www.sumo.dlr.de/userdoc/TraCI/Interfacing_TraCI_from_Python.html

For information about default functions available to users via TraCI, refer to: https://sumo.dlr.de/wiki/TraCI

Function definitions are located in your SUMO_HOME/tools/traci directory. These functions are listed, defined, and their usage explained in files like _vehicle.py, _route.py, etc, as well as the wiki page listed above. The user must only call functions on vehicles that exist. If vehicle1 is spawned at step 10, then the user must wait until step 10 to call functions that operate on vehicle1. The syntax for calling functions from different TraCI libraries is as follows: in order to use a function from _vehicle.py, the user must write: `traci.vehicle.getPosition('vehicle1')`. If a user would like to use a function from _junction.py, they must write: `traci.junction.getPosition('n2')`, and so on.

Provided Scripts

Scripts located in the github have a variety of command line configuration options. If the user would like to run the simulation normally with the gui, execute `$ ./script.py`. If the user would like to run the simulation without the gui, execute `$ ./script.py --nogui`. The script then runs the TraCI control loop, interacting with the simulation, using functions from within the traci/tools function library.

In our example scripts, the ability to define the number of steps you would like the simulation to run, as well as which car you would like to dynamically affect are built in:

```
$ ./script.py [steps] [rogue vehicle]
```

For example, if the user wants to run a simulation for 2500 steps and affect veh3, they would write:

```
$ ./my.py 2500 veh3
```

And it would run an example simulation on our default map with _ vehicles, _ed

Additional Functions Provided with the Patch

Within _vehicle.py are additional functions, written explicitly for usage on rogue cars. Note: All togglers assume user is using default values. If the user is using custom values, they will be overwritten by the toggler.

- rogueToggleFollowDistance('vehID')

    - Toggles car follow distance between default (2.5m) and rogue setting (0.5m). Pass vehicle ID.
- rogueToggleLightRun(‘vehID’)
    - Makes car run stoplights and not respect right of way at intersections. Toggles on and off depending on current status. Pass vehicle ID.
- rogueToggleFollowSpeed(‘vehID’)
    - Toggles car speed factor between default (follow the speed limit) and rogue (double the speed limit). Pass vehicle ID.
- rogueNodeException(‘vehID’, double x, double y)
    - Makes previously declared rogue car respect an intersection and stop for a stoplight. This includes all connected edges that pass into an intersection. Pass vehicle ID, and x, y coordinates of the designated node.
- rogueEdgeException(‘vehID’, double x, double y)
    - Makes previously declared rogue car respect the speed limit along a particular edge. Pass vehicle ID, and x, y coordinates of the connecting points of the desired edge.
- spoof(‘vehID’, ‘route’, typeID = ‘Rogue’, depart =‘triggered’)
    - Makes vehID vehicle spoof 5 different vehicles, 4 of which are invisible. Each vehicle has the ability to have a DSRC device attached to it, making a single vehicle able to have 5 different DSRC devices attached to it. The route is the route that the spoofed vehicles will follow (you should input the route of the original vehicle), the typeID is the type of vehicle you want spoofed, and the depart = ‘triggered’ line should be left the same, because it makes the spoofed vehicles spawn where all other vehicles are designated to spawn.
- pothole(double x, double y)
    - Places a pothole at the given coordinates, which vehicles will avoid and slow down for.

Addition of a DSRC device for collecting Basic Safety Messages

When running simulations using SUMO, you are able to collect vehicle information during a trip using containers called **Devices**. As a vehicle runs from its point of origin to destination, these devices record differents kind of information such as car emissions, trip information, or even battery information. More information on the available devices can be found at https://sumo.dlr.de/wiki/Developer/How_To/Device. By default, the source code files of these devices can be found by navigating to:

$ ~/src/sumo/src/microsim/devices

This folder contains different device modules that can be added to any vehicle via the command line or the .rou.xml file used by the SUMO configuration file, which will be discussed briefly in the **Instantiating the MSDevice_DSRC device** section. One of the contributions of our project was creating a new device module called MSDevice_DSRC.cpp/h whose main functionality is to collect vehicle safety information. This vehicle safety information is based on the SAE J2735 BSM standard so that it can be further developed for Dedicated Short Range Communication applications. For more information on DSRC applications, please refer to https://www.its.dot.gov/itspac/october2012/PDF/data_availability.pdf . Due to the current state of SUMO, not all Basic Safety Message parameters were available so one of our tasks was to create the missing parameters based on the information already provided by SUMO. A summary of the type of information collected and created can be found below under **MSDevice_DSRC Specifications**. You can find MSDevice_DSRC.cpp/h under the devices folder as mentioned in the directory above.

MSDevice_DSRC Specifications

This section will go over the information that the MSDevice_DSRC.cpp/h files collect in detail. Below is a table of the Basic Safety Messages information retrieved by the device. The first table will show the BSM information that was available from SUMO using their getHelper functions. getHelper functions are functions that take a vehicle object as a parameter and can access a vehicles object members such as speed, acceleration, location, etc. The second table below contains BSM information that was not available through SUMO and were created by our team in order to simulate those BSM parameters as closely as possible.

List of BSM attributes available from SUMO source code

Basic Safety Message parameter	Description
Position (lat/long/elev)	Latitudinal and longitudinal coordinates (z-direction support is not currently available in SUMO)

Vehicle length	Vehicle length in meters
Vehicle width	Vehicle width in meters
Vehicle height	Vehicle height in meters
SteeringWheel Angle	Steering wheel angle proportional to the tire angle position
Vehicle identification	Unique ID of the vehicle upon runtime
Vehicle Speed	Current Vehicle speed
Vehicle Acceleration	Current vehicle acceleration
Vehicle Slope Angle (altitude)	Current vehicle slope angle relative to the x-axis
Dsecond	Timestamp DSRC message was went to the Road Side Unit. Time stamp displays date and time with millisecond precision

List of BSM attributes NOT available from SUMO source code

Basic Safety Message parameter	**Description**
Brake system status	Information about the vehicle's braking system. Currently all we have implemented is the detection whether the brakes are ON or OFF based on vehicles speed.
MsgCount	Counter that keeps track of the number of messages sent to a RSU
TemporaryID	A combination of the Vehicle ID and the message count, can be modified
TransmissionState	Information about the vehicles transmission state. Currently a simple mechanism can detect whether a vehicle is PARKED or on DRIVE based on the vehicles speed

Road Side Unit ID	Road Side Unit ID, currently hardcoded
Road Side Unit Signal Strength	Calculates the signal strength between the vehicle and the neighboring Road Side Unit based on location and signal range.

Functions definitions inside MSDevice_DSRC.cpp/h

int getBrakeSystemStatus(double prevSpeed, double currSpeed);

A basic getHelper function that calculates the current brakesystem status status of the vehicle based on the speed of the vehicle at time t and t-1. If the current vehicle speed is less than the previous speed at time t-1, we can assume that the car is slowing down

NOTE: brake system is not currently supported in SUMO so braking status returned is completely based on the changing speed of the vehicle

INPUT: prevSpeed is the speed of the vehicle at the previous timestep t-1
INPUT: currSpeed is the speed of the vehicle at the current timestep t

RETURN: the BrakeSystem status of the vehicle based on previous and current speed

int getTransmissionStatus(double currSpeed);

A basic getHelper function that calculates the current TransmissionStateSystem status of the vehicle based on the speed of the vehicle at time t. If the current vehicle speed is 0, then the car is PARKED. Less than 0 we assume the vehicle is in REVERSE, or if speed is positive, then the vehicle is in the DRIVE state.

NOTE: transmission status is not currently supported in SUMO so the transmission status returned is completely based on the changing speed of the vehicle

INPUT: currSpeed is the speed of the vehicle at the current timestep t

RETURN the transmission status of the vehicle based current speed

int RoadSideUnitDetect(double X_coordinate, double y_coordinate);

Checks if vehicle is within the range of an Road Side Unit currently there is no inter-object communication between a vehicle and Road Side Unit, so as a temporary "hacky" solution is to hardcode the location of a RSU (found inside the additional files) in the simulation and output data to a file when it comes across the vicinity of the RSU.

NOTE: x_coordinate and y_coordinate are the x and y coordinates of the vehicle. This will check the hardcoded #define'd values on lines 65-90 and return whether the vehicle is inside the range of hardcoded values

INPUT: x_coordinate is the current longitudinal coordinate of the vehicle
INPUT: y_coordinate is the current latitudinal coordinate of the vehicle

RETURN: if the vehicle is within the vicinity of an RSU or near none

int RSUSignalStrength(double x_coordinate, double y_coordinate, double x_rsu_coordinate, double y_rsu_coordinate);

Calculates the signal strength of the Vehicle to the RSU using the distance formula. Signal strength is determined by how far a vehicle is from the x, y coordinates of the RSU
INPUT: x_coordinate is the current longitudinal coordinate of the vehicle
INPUT: y_coordinate is the current latitudinal coordinate of the vehicle
INPUT: x_rsu_coordinate is the current longitudinal coordinate of the RSU
INPUT: y_rsu_coordinate is the current latitudinal coordinate of the RSU

RETURN: signal strength of the connection between the vehicle and RSU

bool notifyMove(SUMOVehicle& veh, double oldPos,
double newPos, double newSpeed);

Checks for waiting steps when the vehicle moves, always outputs when the vehicle moves (Function based on MSDevice_Example.cpp/h files)

INPUT: veh Vehicle that asks this reminder.
INPUT: oldPos Position before move.
INPUT: newPos Position after move with newSpeed.
INPUT: newSpeed Moving speed.

RETURN: TRUE (always).

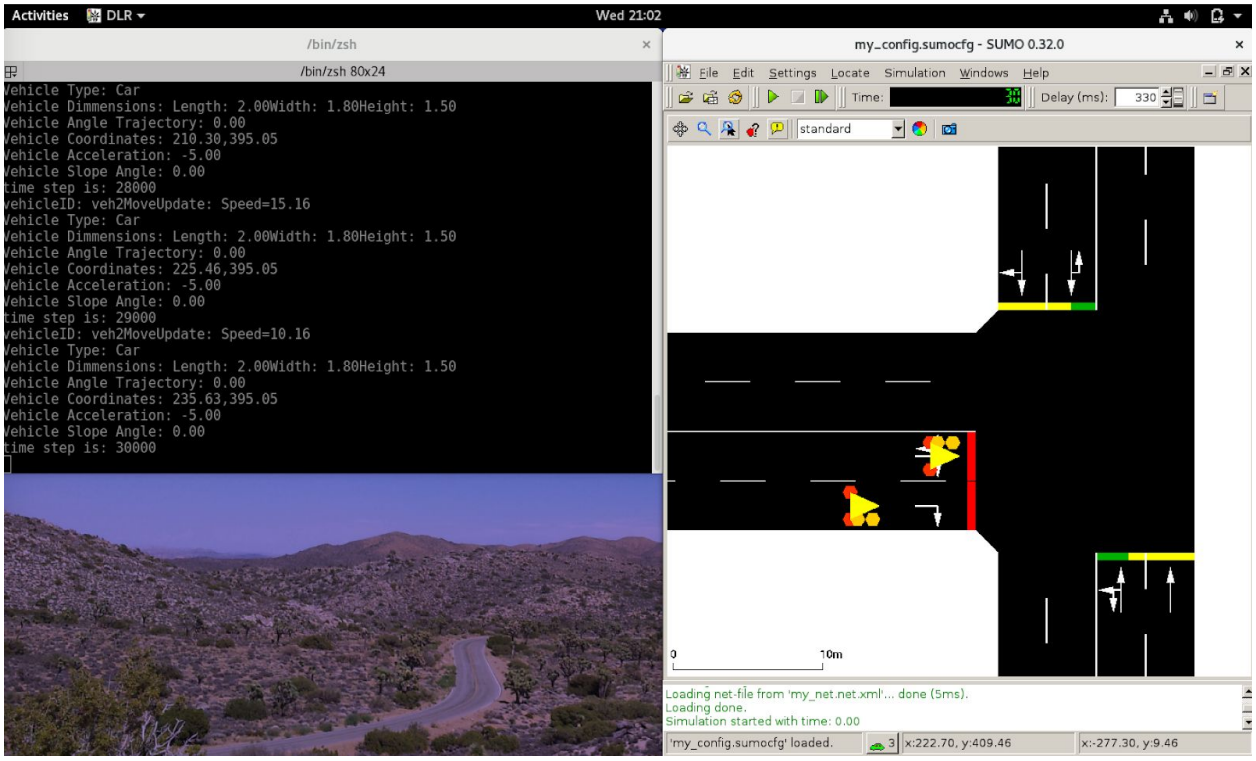
Installing MSDevice_DSRC.cpp/h to your current SUMO directory

There are two methods of installing the DSRC device module that we will present.

In order to use the DSRC device, it must be instantiated on a vehicle, allowing a user to collect BSM data from that specified vehicle. Assuming there is a simulation already created, go to the simulation folder where the sumocfg file is located. There is a .rou.xml file in which you must instantiate the DSRC device for any vehicle. In order to add the DSRC device module on a vehicle, you must go to any vehicle declaration of your choosing and add a new parameter key “has.dsrc.device” and set its value to “true” in order to activate the device during runtime. An example of it is shown highlighted and in bold. This step can be repeated for as many vehicles needed.

```
<vehicle depart="20" id="veh2" route="route2" type="Car">
<param key="has.dsrc.device" value="true"/>
</vehicle>
```

Now the DSRC is activated and will be able to produce BSM information for DSRC applications. You can run a sumo-gui simulation and you will be able to see captured BSM data on an output file as a csv file.



BrakeSystemStatus and TransmissionState were inferred data fields but the rest came from SUMO. Time was created by us. This version listed the time as the current time but in the latest version, we listed time in milliseconds.

MsgCount	TemporaryID	VehicleID	Vehicle Type	Vehicle Length	Vehicle Width	Vehicle Height	Vehicle Wheel	Longitude	Latitude	Vehicle Speed	Vehicle Acceleration	Vehicle Slope Angle	BrakeSystemStatus	TransmissionState	Dsecond
1	veh2_1	veh2	Car	2	1.8	1.5	0	12.1	395.05	10	10	0	BRAKES_OFF	DRIVE	Wed May 1 21:59:46 2019
2	veh2_2	veh2	Car	2	1.8	1.5	0	32.1	395.05	20	10	0	BRAKES_OFF	DRIVE	Wed May 1 21:59:47 2019
3	veh2_3	veh2	Car	2	1.8	1.5	0	62.1	398.35	30	10	0	BRAKES_OFF	DRIVE	Wed May 1 21:59:47 2019
4	veh2_4	veh2	Car	2	1.8	1.5	0	101.394	398.35	39	9.2935	0	BRAKES_OFF	DRIVE	Wed May 1 21:59:47 2019
5	veh2_5	veh2	Car	2	1.8	1.5	0	135.687	398.35	34	-5	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
6	veh2_6	veh2	Car	2	1.8	1.5	0	164.981	398.35	29	-5	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
7	veh2_7	veh2	Car	2	1.8	1.5	0	190.142	395.05	25	-4.13208	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
8	veh2_8	veh2	Car	2	1.8	1.5	0	210.303	395.05	20	-5	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
9	veh2_9	veh2	Car	2	1.8	1.5	0	225.465	395.05	15	-5	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
10	veh2_10	veh2	Car	2	1.8	1.5	0	235.626	395.05	10	-5	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
11	veh2_11	veh2	Car	2	1.8	1.5	0	240.788	395.05	5	-5	0	BRAKES_ON	DRIVE	Wed May 1 21:59:47 2019
12	veh2_12	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	-5	0	BRAKES_ON	PARK	Wed May 1 21:59:48 2019
13	veh2_13	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	-0.161417	0	BRAKES_OFF	PARK	Wed May 1 21:59:48 2019
14	veh2_14	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	0	0	BRAKES_OFF	PARK	Wed May 1 21:59:48 2019
15	veh2_15	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	0	0	BRAKES_OFF	PARK	Wed May 1 21:59:48 2019
16	veh2_16	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	0	0	BRAKES_OFF	PARK	Wed May 1 21:59:48 2019
17	veh2_17	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	0	0	BRAKES_OFF	PARK	Wed May 1 21:59:48 2019
18	veh2_18	veh2	Car	2	1.8	1.5	0	240.949	395.05	0	0	0	BRAKES_OFF	PARK	Wed May 1 21:59:48 2019

When cars are in proximity of a certain RSU, the BSM data only during that time period gets printed out. Here is an example.

MsgCount	TemporaryID	VehicleID	Vehicle Type	Vehicle Length	Vehicle Width	Vehicle Height	Vehicle Wheel Angle	Longitude	Latitude	Vehicle S
548	veh1a_548	veh1a	Rogue	2	1.8	1.5	0	304.607	598.35	14
557	veh1a_557	veh1a	Rogue	2	1.8	1.5	0	319.607	598.35	15
558	veh1b_558	veh1b	Rogue	2	1.8	1.5	0	302.106	598.35	14
566	veh1a_566	veh1a	Rogue	2	1.8	1.5	0	335.607	598.35	16
567	veh1b_567	veh1b	Rogue	2	1.8	1.5	0	317.106	598.35	15
575	veh1a_575	veh1a	Rogue	2	1.8	1.5	0	352.607	598.35	17
576	veh1b_576	veh1b	Rogue	2	1.8	1.5	0	333.106	598.35	16
577	veh1_577	veh1	Rogue	2	1.8	1.5	0	314.605	598.35	15
584	veh1a_584	veh1a	Rogue	2	1.8	1.5	0	370.607	598.35	18
585	veh1b_585	veh1b	Rogue	2	1.8	1.5	0	350.106	598.35	17
586	veh1_586	veh1	Rogue	2	1.8	1.5	0	330.605	598.35	16
587	veh1c_587	veh1c	Rogue	2	1.8	1.5	0	307.104	598.35	14
593	veh1a_593	veh1a	Rogue	2	1.8	1.5	0	389.607	598.35	19
594	veh1b_594	veh1b	Rogue	2	1.8	1.5	0	368.106	598.35	18
595	veh1_595	veh1	Rogue	2	1.8	1.5	0	347.605	598.35	17
596	veh1c_596	veh1c	Rogue	2	1.8	1.5	0	322.104	598.35	15

Limitations of MSDevice_DSRC.cpp/h